J Real-Time Image Proc (2018) 15:123–159
https://doi.org/10.1007/s11554-016-0588-9

SPECIAL ISSUE PAPER

Real-time hardware–software embedded vision system for ITS smart camera implemented in Zynq SoC

Tomasz Kryjak[1] · Mateusz Komorkiewicz[1] · Marek Gorgon[1]

Received: 8 June 2015 / Accepted: 21 March 2016 / Published online: 4 May 2016

Abstract The article demonstrates the usefulness of heterogeneous System on Chip (SoC) devices in smart cameras used in intelligent transportation systems (ITS). In a compact, energy efficient system the following exemplary algorithms were implemented: vehicle queue length estimation, vehicle detection, vehicle counting and speed estimation (using multiple virtual detection lines), as well as vehicle type (local binary features and SVM classifier) and colour (k-means classifier and YCbCr colourspace analysis) recognition. The solution exploits the hardware–software architecture, i.e. the combination of reconfigurable resources and the efficient ARM processor. Most of the modules were implemented in hardware, using Verilog HDL, taking full advantage of the possible parallelization and pipeline, which allowed to obtain real-time image processing. The ARM processor is responsible for executing some parts of the algorithm, i.e. high-level image processing and analysis, as well as for communication with the external systems (e.g. traffic lights controllers). The demonstrated results indicate that modern SoC systems are a very interesting platform for advanced ITS systems and other advanced embedded image processing, analysis and recognition applications.

Keywords Intelligent Transportation Systems · Hardware-software image processing (Zynq SoC) · Vehicle queue length estimation · Vehicle detection · Vehicle type and colour recognition

1 Introduction

The use of vision systems in vehicle traffic analysis and control (so-called intelligent transportation systems—ITS) is becoming more and more widespread. It is evidenced by the increasing number of cameras installed near the roads. They are used to:

- analyse and control traffic at intersections,
- monitor traffic congestions,
- detect unusual events (e.g. collisions or accidents),
- estimate travel time between two cities,
- measure average speed on a given road section,
- measure vehicle speed,
- detect red light crossing,
- collect toll (car parks, motorways).

The first of the mentioned applications seems to be especially important. A vision system mounted at the intersection can provide a range of relevant information. Firstly, it can be used to estimate the car queue length, i.e. detect the presence of vehicles in certain locations. This can be achieved with the use of so-called virtual induction loops or virtual detection areas. Alternative solutions such as induction loops, passive magnetic sensors or pneumatic

The work presented in this paper was supported by AGH Univeristy of Science and Technology project number 15.11.120.476 (first) and 11.11.120.612 (second and third author).

✉ Tomasz Kryjak
tomasz.kryjak@agh.edu.pl

Mateusz Komorkiewicz
komorkiewicz@gmail.com

Marek Gorgon
mago@agh.edu.pl

[1] Department of Automatics and Biomedical Engineering, Faculty of Electrical Engineering, Automatics, Computer Science and Biomedical Engineering, AGH University of Science and Technology, al. Mickiewicza 30, 30-059 Krakow, Poland

tubes have a common drawback. Their installation and maintenance in case of a failure require interference with the road surface, which can be quite costly. In addition, the obtained information is only binary (vehicle/no vehicle). On the other hand, these solutions are quite reliable and not very sensitive to external conditions (time of the day, weather, etc.).

A vision-based system also allows for: vehicle counting, vehicle mean speed estimation, rough classification of vehicles (e.g. bikes/motorbikes, cars, minibuses, buses, trucks) and rough colour estimation (only basic colours: white, black, red, green, blue, etc.). Furthermore, it is possible to track vehicles between intersections (using the plate number or other features), detect abnormal situations (accidents, breakdowns, etc.) or even to exactly classify the vehicles (make and model) [25]. Moreover, the human operator of the ITS system should be able to download images from the camera and accurately assess the current situation—for example, using a web-based access interface.

However, it is also noteworthy to point out certain disadvantages of the vision-based system. First, these solutions are often affected by lighting and meteorological conditions. Good examples are: night-time, deep shadows, heavy rain, snowfall or fog. Great challenges are also posed by: the huge variety of vehicles (different sizes and shapes) and the limited computational power (real-time image processing vs. low power consumption).

In the case of vision systems, the required calculations can be realised in two variants. In the first, the video stream from cameras mounted at intersections is transmitted to the surveillance centre, where it is subjected to manual or automatic analysis. The main disadvantage of this approach is the need for very high bandwidth communication infrastructure. In the second approach, the smart camera [8] concept is used. In this case, image processing and analysis are carried out immediately after image acquisition and there is no need to transmit every frame to other components of the system. Usually, the output contains only simple data such as the vehicle queue length or the number of detected vehicles (so-called meta-data stream). Of course, the smart camera should also allow to access the raw video stream, as this can be useful in the analysis of unusual situations or debugging the system.

When designing a smart camera, a very important issue is the choice of the hardware computing platform. There are solutions based on general-purpose processors (GPP), digital signal processors (DSP) and reconfigurable devices (FPGA—field programmable reconfigurable arrays). For performing real-time image processing and analysis with relatively low energy usage the third platform seems to be very attractive. In recent years it has been proved that reconfigurable systems can handle many vision algorithms such as various filtration methods, complex background modelling and foreground object segmentation. Further examples are optical flow computation, tracking and object classification systems (e.g. pedestrian detection) [7]. Many image processing systems implemented in FPGAs involve the pipeline data processing approach, where the pixel stream passes through different computing elements.

However, for some complex vision algorithms the pipeline implementation proved quite cumbersome or even impossible. A good example is the region growing segmentation, which requires an unpredictable number and order of pixels accesses. In such cases, the use of a general purpose processor system is a much more convenient solution. Modern FPGAs allow to use a so-called soft-processor (MicroBlaze from Xilinx, Nios from Altera), but these solutions have quite limited computing performance. In 2012 Xilinx introduced the Zynq heterogeneous platform, which is a combination of FPGA logic resources and a dual-core ARM processor [71]. The portfolio consist of the Zynq 7000 SoC series and Zynq UltraScale+ MPSoC series. The first one contains an ARM Cortex-A9 dual-core processor, the second a quad-core ARM Cortex-A53 processor, a dual-core ARM Cortex-R5 processor and an ARM Mali-400MP GPU (graphics processing unit).

Similar devices are also available from Altera [2]. They are called Altera SoC. The portfolio consists of Cyclone V SoC, Arria V SoC, Arria 10 SoC and Stratix 10 SoC. The first three devices contain an ARM Cortex-A9 dual-core processor and the last a 64-bit quad-core ARM Cortex-A53.

The solution has gained some interest in the embedded image processing community. In the end of year 2015 over 30 SoC-based papers were available in the most popular databases. They cover different topics:

- image filtering [16],
- feature extraction [28, 60],
- optical flow computation [42],
- road sign recognition [54],
- driver awareness monitoring system [56],
- face detection [23, 77],
- stereovison system [10],
- object detection and tracking [49],
- advanced driver assistance systems [59].

In this paper the concept of using the Zynq SoC device in an embedded smart camera for intelligent transportation systems is considered. To the best of our knowledge this is the first reported approach of this type.

On a single hardware–software system—reprogrammable logic and ARM processor system with PetaLinux [70] operating system—the following algorithms were implemented:

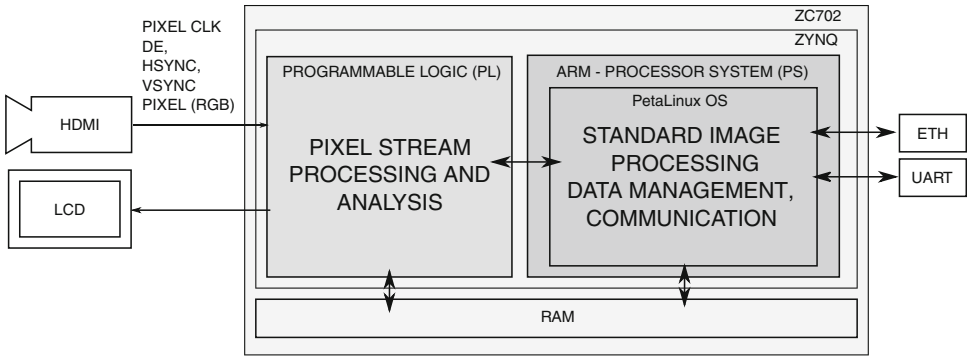

Fig. 1 Concept of the proposed embedded hardware–software system

- vehicle queue length estimation,
- vehicle detection, counting and speed estimation,
- vehicle type recognition,
- vehicle colour recognition.

The main contributions of the paper can be summarised as follows:

- the concept of using the heterogeneous (hardware–software) Zynq SoC device for ITS smart-camera, which allows to obtain an effective, low-power computing platform,
- evaluation of this concept by implementing sample algorithms used in ITS smart cameras,
- the proposal of a new and robust vehicle detection algorithm customized for a hardware–software system,
- the proof that a Zynq-based smart camera allows real-time image processing of a 720 × 576 @ 50 fps pixel video stream.

The paper is organized as follows. First, the general concept of an embedded hardware–software image processing system is discussed in Sect. 2. Then, in Sect. 3 the prior-mentioned algorithms used in ITS are reviewed. A general overview of the proposed system is provided in Sect. 4. Then particular modules are described: vehicle queue length estimation (Sect. 5), vehicle detection and counting (Sect. 6), vehicle type and colour recognition (Sect. 7). In Sect. 8 modules (so-called global), which are common for the image processing system are discussed. Integration of the system on the Xilinx ZC 702 development board, as well as evaluation results are presented in Sect. 9. The article ends with further research discussion and a short summary.

2 Concept of an embedded hardware–software vision system

In this work, an embedded hardware–software vision system based on the heterogeneous Zynq platform is discussed. Its architecture is presented in Fig. 1.

It consists of the following devices:

- a video camera with HDMI output (source of the video stream),
- a computing platform (heterogeneous Zynq device),
- an evaluation board containing a Zynq device, RAM and peripherals (ZC 702 board from Xilinx),
- a display device (LCD monitor).

The source of the video stream is a digital camera with high-definition multimedia interface (HDMI) output.[1] In this case the user logic receives the following signals:

- pixel clock (PIXEL CLK),
- data validity signal (DE),
- horizontal synchronization signal (HSYNC),
- vertical synchronization signal (VSYNC),
- pixel data (e.g. 24 bits per pixel in RGB format).

The used Zynq computing platform consists of programmable logic resources (PL) and a processing system (PS). The heterogeneous system is a part of an evaluation board, which contains also the required input/output interfaces and external RAM memory. In the presented solution, it is also possible to visualize image processing results via an HDMI output. Furthermore, image analysis results (metadata) are available through Universal Asynchronous Receiver Transceiver (UART), Ethernet or a simple web service.

In this section the pipeline data processing concept is presented. In addition, the assumptions that were adopted during distributing the computational task between PL and PS are discussed. Furthermore, the choice of the operating system for the PS is explained.

2.1 Pipeline data processing

A hallmark of the used video source is the stream data transmission method. It was implemented in analogue

[1] In general, a tighter integration, i.e. direct communication between the complementary metal-oxide semiconductor (CMOS) or charge coupled device (CCD) vision sensor and the reprogrammable logic is also possible.

video transmission systems (i.e. CCIR TV). Today, this method is also used in digital standards, i.e. HDMI.

When the image acquisition phase is complete, the camera transmits data pixel by pixel, line by line, frame by frame. These data are not compressed, the transmission does not require the use of complex protocols and it is not necessary to buffer the image frame when forming the signal. The disadvantages of this solution are, however, the high signal bit rate (related to the high clock frequency of the synchronous data bus) and the limited length of the cable. Therefore, the tight integration of image acquisition and processing systems is highly desirable. This is often referred to as the smart-camera concept [8].

Processing and analysis of video stream data in a pipelined system is a well-known and often used technique. The first implementations of basic image processing operations were proposed over 20 years ago in [4, 64] or [19]. The development of the FPGA technology enabled to use these devices for image analysis and recognition [14, 27]. Furthermore, the study [20] showed that pipelined image processing systems can achieve linear acceleration. Outstanding computing performances of parallel-pipelined modules for the Horn–Schunck optical flow algorithm were demonstrated in papers [31, 35].

Depending on the used algorithms, it may be necessary to gather a local context of the image. This can be done easily with so-called delay lines (e.g. for image filtering). In some cases, it is necessary to temporary cache the full image frame. Examples include two-pass connected component labelling [27], optical flow computation [31] or foreground object segmentation [34]. Nevertheless, the data temporarily stored in the buffer are transmitted again to the rest of the system in the form of a video stream.

The most important parameter of a pipeline system is the data processing frequency. To ensure real-time operation, the frequency of all used hardware modules (processing elements) should be equal or greater than the so-called pixel clock (i.e. the frequency of pixel propagation). Video stream parameters such as the number of pixels in a single frame (i.e. image resolution), the number of frames per second (fps) and pixel representation (e.g. 8-bit greyscale or 24-bit RGB) determine the required frequency of the system. For example, for resolution 720×576 @ 50 fps it equals 27 MHz and for 1920×1080 @ 50 fps—148.5 MHz.[2] Thus, the key challenge when designing a pipeline processing system is to create a computing architecture that meets the above requirement. The computing performance of the pipeline system also depends on the complexity of the used algorithms. In [20] it has been shown that the performance of a pipeline solution depends on the number of operations performed for a single pixel and pixel propagation frequency.

In the defined pipeline architecture, the suspension of processing is not allowed, as data are transmitted continuously. The basic data unit in this system is a single pixel. Therefore, it is referred to as fine-grain.

The above-described fine-grain pipeline data processing system is essentially different from a typical software solution implemented on a general purpose processor (GPP). In the latter case the basic data unit is a single frame. Therefore, the system is described as a course-grain. A typical software video processing application operates in three steps: image acquisition and storing the image in the input memory (from a camera or hard disk), calculations and saving of output data. Although single pixel processing (i.e. `for` loop over the image) is often used, in many cases it could be parallelized. The data transfer between the GPP and RAM is usually efficient enough to obtain larger image chunks and process them in parallel. An example would be pedestrian detection in several areas of the image executed in separate threads (each on a separate processor core).

The real-time requirement for coarse-grain processing is defined in a different way than for the fine-grained one. Here, the overall calculation time cannot exceed the time between the acquisition of two consecutive frames (e.g. 1/50 s for 50 fps). However, if the above condition is not met, it is usually possible to reduce the processing rate (e.g. from 50 to 33 fps). This solution is acceptable in non-time critical vision systems.

In a pipeline processing system such a situation is not possible. The execution time limit for each computation step results from the pixels propagation frequency. If this time is exceeded, even in one element of the system, the obtained results are completely wrong and useless. In practice, the designer verifies the maximum pixel clock frequency at which the system operates properly (using timing analysis tools and hardware verification). Then, the maximal allowable video stream parameters (resolution and frame rate or bit rate) can be selected and verified.

The pipeline data processing architecture implies certain limitations. For example, algorithms involving operations that are dependent on pixel order in the data stream are impossible to implement—e.g. region growing segmentation, where the data access pattern depends on image content. In many other cases, it is necessary to impose some additional restrictions such as the maximum number of objects in connected component labelling. However, despite the mentioned drawbacks, the pipeline scheme is a very attractive solution. First, because it is consistent with the data sending method from the image sensor, and therefore frame buffering between successive operations is

[2] Due to the presence of front porches, sync pulses and back porches the pixel clock frequency is given by the formula: (horizontal resolution + front porch + sync pulse + back porch) × (vertical resolution + front porch + sync pulse + back porch) × fps.

not required. Second, as already mentioned, the linear acceleration in a pipeline system [20] allows to achieve very high computing performance (operations per second) and energy efficiency (operations per watt) [31].

Thus, if the main requirement of an embedded vision system is real-time processing of a video sequence (especially high definition), the pipeline data processing is particularly recommended. Therefore, the majority of calculations should be implemented in hardware resources (PL). However, there are some cases when designing a hardware module is not possible or very complicated and, therefore, not justified. Then, a software processing system with random assess to each pixel should be used.

The integration of both types of architectures proved to be feasible in the heterogeneous Zynq device. The fine-grain part can be implemented efficiently in reprogrammable resources (PL). The coarse-grain processing can be realised on the ARM processor cores.

The above-discussed assumptions and fine-grain and coarse-grain architectures were used in the proposed vision system. The majority of image processing algorithms were implemented in a pipeline manner in reprogrammable resources. The ARM processor was used only for operations that would be difficult to implement in PL, i.e. complex image processing, Ethernet communication, database, web server, etc. It should be also noted that this approach allowed to limit the number of transfers between the PL and PS. Essentially, the transfer was only one directional from PL to PS.

Finally, it should be noted that the presented method of using heterogeneous devices for image processing systems is not the only one possible solution. In many real-life cases, the input data do not origin directly from a video source, but is transferred from a host PC via PCI-X bus or received from a camera with Ethernet (e.g. GigE) or USB interface. The latter solutions often involve intra- or interframe video compression. Thus, the basic data chunk is not a pixel, but a part or the entire image frame. In this situation, a coarse-grain or middle-grain approach is required. However, some of the computing intensive tasks (so-called bottlenecks) can be transferred by the processor to the accelerating modules in PL. With such assumptions, before splitting the computing tasks between hardware and software, a thorough code profiling should be performed to determine all bottlenecks. In the above-described approach, the GPP is considered as the primary computation platform. Nevertheless, such a system architecture and design methodology is completely opposite to the chosen by the authors of this paper.

2.2 Operating system selection for the ARM processor

After specifying the role of the ARM processor in the proposed vision system, it is possible to choose the best suited operating system (OS). For the Zynq platform the following options are available: the so-called bare metal (without OS), Linux OS, real-time operating system (RTOS) and Android OS. The simplest solution is a bare metal application. It is easy to implement and efficient. However, the basic features of an OS like multitasking, applying libraries (e.g. OpenCV), easy communication via Ethernet are not supported. Moreover, this solution limits and complicates the potential future development of the system.

Another solution is the use of the Linux operating system. Several distributions, both free and commercial, are available for the Zynq platform. The basic, supported by Xilinx, is PetaLinux [70]. Others include: Arch Linux Distribution, Denx ELDK, ENEA Linux, MontaVista Linux, SYSGO ELinOS, Timesys LinuxLink, Wind River Linux and Xillinux.

The third possibility is the use of a real-time operating system, for example, FreeRTOS. In the presented vision system, this solution was not considered, as no time-critical applications were assigned to the processor. Furthermore, it is possible to run Android OS on the Zynq, however, this option has no advantages over Linux, and therefore was also not considered.

From the above-listed solutions, the PetaLinux system was chosen because it contains all the necessary drivers, tools (boot loaders, device drivers, etc.) and a well-developed community support. In addition, Xilinx has prepared a Board Support Package (BSP) for the ZC 702 platform used in the experiments. It contains a properly configured, ready-to-build system distribution. This allowed to implement the described vision system easier and quicker. Furthermore, the required functionalities: communication via SSH, FTP, SFTP, web server or running the OpenCV library (for performance evaluation of the software model) were almost instantly available. The use of another distribution would involve a fairly tedious configuration process, which is beyond the scope of this research.

It is also worth mentioning that for more advanced applications both ARM processing cores could be used. For example, a configuration with Linux on one core, and RTOS or bare-metal on the second is possible. In the context of future development of the considered application the Linux + RTOS option could be particularly interesting. The RTOS would be responsible for tasks directly related to traffic light control and the Linux for less time-critical functions like statistics and communication.

3 Algorithms used in smart cameras for ITS

In this section the most widespread ITS vision algorithms are discussed: vehicle queue length estimation, vehicle detection and counting, vehicle speed estimation, as well as vehicle type and colour recognition.

3.1 Vehicle queue length estimation

The problem of vehicle queue length estimation has been widely described in scientific papers. This issue is very significant, because information about the queue length can be almost directly applied in an intelligent traffic light controller.

Algorithms In the work [55] the analysis area (i.e. the intersection) was divided into several separate blocks (called ROI—regions of interest). The width of a single block corresponded to the width of the roadway and the height to the average length of a typical car. The authors of the algorithm made use of vehicle queue forming properties—the cars stop one by one beginning from the marked stop line. In the first stage, after the traffic light turns red, the ROIs closest to the camera were analysed. When a stopped vehicle was detected, the queue counter was increased and the next block was analysed. The counter was set to zero, when the first car started to move. This approach saved computing resources, as only two blocks had to be analysed for a single lane (queue head and tail).

The most important element of the algorithm was the procedure that allowed to determine whether the car was located within the given block. The authors proposed a two-stage approach. In the first step, the number of edges (Sobel edge detector) was used to detect a vehicle. Typically, the vehicle had much more edges than the road surface. However, the detection in areas where a deep shadow cast by trees or other objects nearby the road was present sometimes provided incorrect results. Therefore, in the second stage so-called dark areas were detected. The required binarization threshold was calculated as a certain percentage of the mean brightness in the given ROI.

The two-step approach described above allowed to determine the status of the block. The algorithm was evaluated on 45 short sequences. In total over 32,000 different ROIs were tested and 99.9 % accuracy was reported. The only errors were caused by the presence of large vehicles which contained a small number of edges and, therefore, were not detected correctly.

A similar solution was described in the work [75]. It was based on movement detection (consecutive frame differencing) and vehicle detection (entropy and edge based) in disjoint blocks. Also a comparable approach was used in the paper [76]. The authors also proposed a queue severity index and the methodology for selecting all thresholds used in the algorithm. This system was extensively evaluated during a 6-month period.

A slightly different approach was applied by the authors of the paper [1]. The vehicle detection was based on corner features (Harris algorithm). This resulted from the observation that the road surface is generally uniform and vehicles usually have many corners. Additionally, the information about movement was obtained using consecutive frame differencing. The queue length was determined by counting corners on static parts of a given lane. The authors also used a perspective correction based on homographic transformation, which allowed to estimate the queue length in meters. Extensive tests in different conditions confirmed the effectiveness of the proposed solution.

A different solution was proposed in [72]. The camera was placed not on or above the traffic lights but in such a way so that the cars were visible from the back. The rear parts of the vehicles were detected using Haar features and AdaBoost cascade classifier. Also movement and edge detection were used. The lane markings were eliminated using a pattern matching-based approach. The queue length was determined using two moving windows—one for the head and second for the tail of the queue.

The author of the work [51] applied linguistic variables and fuzzy set theory to detect the presence of a vehicle in a given area. This method involved the analysis of local context size of 7×7 pixel size used to determine the relationship between the central pixel and the corners. The relationships were then described with the use of fuzzy sets. The detection was based on counting and thresholding the found attributes. The method was tested on more than 20 h of recordings and proved to be robust. The author emphasizes the possibility of an efficient hardware implementation.

Embedded implementations There are also several articles describing embedded implementations of ITS systems.

In the article [63] a hardware module for queue length detection was described. A DSP was used as the computation platform. The algorithm was based on the thresholding of the input image using the Otsu method [47] for determining the threshold. During segmentation objects lighter and darker than the road surface were detected. The resulting object mask was filtered and analysed to estimate the queue length.

In the study [74] also a DSP-based embedded vision system was presented. The analysis was carried out in disjoint blocks. Consecutive frames differencing and foreground object detection (using the Gaussian Mixture Models method) was used for motion detection. Vehicle presence detection was based on edge analysis (morphological edge detector). To reduce the computational complexity, motion detection was performed only for the beginning and end of the queue and the presence detection only for the end of the queue. Tests made in different weather conditions during a two weeks period showed high efficiency of the solution.

Summary The above-described algorithms and their implementations are summarized in Table 1. The following parameters were considered:

Table 1 Queue length estimation—summary

Work	Platform	ROI	Detection algorithm	Evaluation	Remarks
[55]	GPP	Block	Edge, so-called dark areas	45 sequences—99 % accuracy %	–
[75]	GPP	Block	Entropy, edge	Acc. not provided	–
[76]	GPP	Lane	Horizontal edges, consecutive frame differencing	Acc. not provided, 6 months of evaluation	–
[1]	GPP	Lane	Corners, consecutive frame differencing	Acc. not provided	Result analysis method difficult to implement in a pipeline vision system
[72]	GPP	Lane	Edge, movement, Harr features, AdaBoost classification, pattern matching	Acc. not provided	Algorithms difficult to implement in a pipeline vision system
[51]	GPP	Block	7×7 local context analysis, fuzzy sets	Less than 1 % errors (11 % during night-time), 20 h video	–
[63]	DSP	Lane	Ots'u thresholding	Acc. not provided	–
[74]	DSP	Block	Consecutive frame differencing, foreground object segmentation (GMM), edge	Acc. not provided, 2 week period evaluation	–
[61]	FPGA	Lane	Edge, entropy	–	Paper in Chinese

- Platform—the used computing platform,
- ROI—region of analysis (whole image, a single lane or a certain part of the lane),
- Detection algorithm—features used for vehicle presence detection,
- Evaluation—available information about evaluation, i.e. used dataset and reported accuracy,
- Remarks.

Most of the listed solutions operate on image parts (blocks) corresponding to a single vehicle. Movement detection is based on consecutive frame differencing. Vehicle presence detection is mainly based on edges, sometimes supported by entropy or dark areas analysis.

Accuracy comparison of the presented solutions is difficult. The ITS vision community did not develop a single video sequence database to evaluate this type of algorithms.[3] Moreover, the authors test their system in real-life conditions for a long period of time (weeks, months). Obviously, this is an excellent approach, but it makes the comparison even harder, as repeating the experiment would require at lot of computations and data storage.

3.2 Vehicle detection

Algorithms Vehicle detection can be performed using foreground object or moving object segmentation. In the first case, objects are usually extracted using the differential image between the current frame and the so-called background model. This approach has been used, among others, in works [24, 11, 17, 53, 3]. Unfortunately, this solution has a number of drawbacks that hinder its practical application in traffic monitoring. These are: low resistance to camera jitter (it is usually necessary to implement some kind of jitter compensation algorithm), difficulties in initializing and reinitializing the background model properly (especially in the presence of heavy traffic), high sensitivity to shadows and sudden illumination changes (e.g. reflections of car lights on the road). In addition, the specific conditions present at an intersection cause that background elements (i.e. carriageway) are obscured by cars waiting for the green light for extended periods of time. This significantly hinders the background update procedure and causes many segmentation errors.

Moving object detection using optical flow or, in a simplified case, consecutive frame differencing, was applied to vehicle segmentation in the work [18]. The solution helps to eliminate some of the disadvantages of background subtraction—e.g. sensitivity to camera movement or problems with maintaining the correct background model. On the other hand, it only allows to segment vehicles that are moving. This greatly complicates the analysis of the situation on an intersection. Furthermore, the obtained object masks often require complex post-processing, since the optical flow field for homogeneous areas is usually incorrect (e.g. division of a large uniform object). In the literature more advanced solutions such as 3D deformable models [50] were also described. However, the computational complexity limits their usage in embedded devices.

A very interesting approach, that is frequently used in recent research papers, are virtual detection lines (VDL)

[3] It is worth mentioning that such solutions exist for multiple video processing topics. Examples are: changedetection.net for foreground object segmentation, http://vision.middlebury.edu/stereo/data/ for stereovision or http://vision.middlebury.edu/flow/ for optical flow.

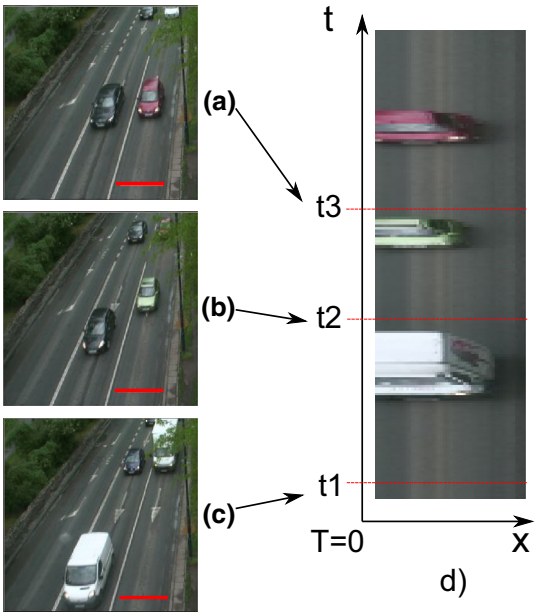

Fig. 2 The idea of using virtual detection lines (VDL) and time-spatial images (TSI). **a** Frame at time $t3$, **b** frame at time $t2$, **c** frame at time $t1$, **d** vehicles on the TSI, x-axis—spatial, t-axis—time [33]

and time-spatial images (TSI). The idea is presented in Fig. 2. The basis is a virtual detection line (VDL) located on a given part of the road (red line). In each frame, the pixels on the VDL are stored in a buffer and form the time-spatial image. The TSI image contains information about vehicles width (x-axis) and vehicle size/speed (t-axis). This can be used to implement vehicle counting, speed estimation or even classification [41, 73].

In the literature, two approaches are presented. The TSI image is generated from the object mask obtained using background subtraction [73] or directly using the raw image from the camera [41]. In the second case, edge detection (e.g. Canny algorithm) followed by some morphological post-processing (closing, filling holes) is often applied. This allows to obtain masks of individual objects (vehicles). It is also possible to integrate the results from several VDL/TSI, which can improve the reliability of the system [41]. An important feature of this approach is the relatively low computational complexity due to processing only a rather small portion of the image. What is more, it allows to detect moving and stopped vehicles (with proper TSI image analysis). Additionally, a system with many VDLs can be easily implemented in a parallel computing architecture, such as FPGA or heterogeneous SoC devices.

Embedded implementations Hardware implementations of vehicle detection algorithms have been described in several papers. One of the first works [21] used a relatively simple background model and the SAD algorithm. To maintain a proper background model, the update was performed only when no vehicles were detected at the specific location. The system was implemented in Handel-C HLS language [39] and PixelStream library [40]. It was evaluated on RC300E platform with a Virtex II FPGA device. It allowed to process 25 frames with 786 × 576 pixels resolution per second.

In the paper [38] a rather simple vehicle motion detection algorithm was reported. It was based on foreground object detection. The module was implemented for a Cyclone II FPGA device. No data about performance were provided. A similar system was also presented in the work [9].

In the paper [57] a hardware implementation of a traffic analysis algorithm was presented. The segmentation was based on background modelling and subtraction (short- and long-term background models), supplemented by shadows and light reflections detection. The application allowed also to measure the speed of vehicles. To increase reliability, a geometric transformation of the image was applied. The system was evaluated on a Virtex 4 FPGA and allowed to process 32 frames with a resolution of 128 × 128 per second. The correct detection rate of the system given by the authors was 90 % at day and 56 % at night (100 cars in each test).

In articles [66, 67] extended versions of the above-described system were presented. Edges were added to the background model, the generalized Hough transform was applied and object tracking based on a binary mask was implemented (executed on a CPU). A very valuable part of the work was its practical verification in urban conditions. A total number of 26 nodes were used (22 based on FPGA and four on ASIC). The authors reported an accuracy of 93 % at a sunny day, 83 % at a cloudy day and 63 % at night (100 cars in each test).

Summary The above-described algorithms and their implementations are summarized in Table 2. The following parameters were considered:

- Platform—the used computing platform,
- Detection method—the used vehicle presence detection approach,
- Evaluation—available information about evaluation, i.e. used dataset and reported accuracy,
- Remarks.

The vast majority of the analysed methods were based on foreground object segmentation and background subtraction. It is worth noting that they were primarily designed for vehicle detection and counting on highways, i.e. without stopped vehicles. In the case of an intersection, especially with high intensity of traffic, the correct background model update is quite difficult. Therefore, approaches without background subtraction seem to be a more promising solution, especially the VDL and TSI proposed

Table 2 Vehicle detection—summary

Work	Platform	Detection method	Evaluation	Remarks
[24]	GPP	Background subtraction	Four highway sequence, acc. 70 %	Advanced shadow elimination procedure
[11]	GPP	Background subtraction	30 min. highway sequence, acc. not provided	–
[17]	GPP	Background subtraction, detection verification by pyramidal HOG + SVM	Four sequences on highway acc. not provided	Also tracking
[53]	GPP	Background subtraction	Highway sequence (3400 frames 76 vehicles), 97.37 % accuracy	Also tracking
[3]	GPP	Background subtraction (GMM)	Five highway image sequences, in favourable conditions up to 100 % accuracy	Also shadow elimination
[18]	GPP	Optical flow (Horn-Schunck)	Intersection sequence, almost aerial view, 95.4 % accuracy	–
[50]	GPP	Background subtraction, deformable 3D model	267 sequences with 3074 vehicles, 100 % accuracy	–
[41]	GPP	Multiple VDLs and TSIs, vehicle detection on TSI based on Canny edge detection	Sequences from Dhaka Bangladesh and Suwon Korea, acc. not provided	Quite complicated logic to handle occlusions
[73]	GPP	Background subtraction, VDL, TSI	Sequences Highway 1, 2 (acc. 89.2 %) and four others (each 50 min) (acc. 97.2 %)	Also shadow elimination
[21]	FPGA	Background subtraction	Acc. not provided	–
[38]	FPGA	Background subtraction	Acc. not provided	–
[9]	FPGA	Background subtraction	Sequence with 247 vehicles, acc. not provided	–
[57]	FPGA	Background subtraction	Day (acc. 90 %) and night (acc. 50 %) seq. with 100 cars each	Shadow and light reflection detection
[67]	FPGA	Background subtraction	Sunny day (acc. 93 %), cloudy day (acc. 83 %) and night (acc. 63 %) seq. with 100 cars each	Shadow and light reflection detection, tracking

in [41]. The vision systems were evaluated on different sequences. In should also be noted that in favourable conditions some systems achieve 100 % detection accuracy.

3.3 Vehicle type recognition

Algorithms The issue of vehicle classification can be divided into two relatively distinct subproblems. The first is the so-called mark and model recognition. It is quite difficult for several reasons. Firstly, modern vehicles from different manufacturers are quite similar to each other, at least within one segment (with a few exceptions). Secondly, the recognition usually is based on logotype analysis, but this is often changed or modified and also placed at various locations. Finally, other features such as traffic lights (front or back) or grille are usually "hard" to describe. There are a few scientific papers covering this topic: [5, 12, 37]. They use the proven histogram of oriented gradients (HOG) and support vector machines (SVM) approach introduced for pedestrian detection in the work [15].

Make and model recognition requires an high-quality input image. Unfortunately, such an image cannot be obtained using typical cameras mounted at intersections. In a standard setup (camera above the road surface) and 720×576 pixel resolution the logotype size is only a few pixels. Therefore, a large part of ITS vision systems enables only a rough vehicle classification into several groups: motorbikes (bikes), passenger cars (hatchback, sedan, estate, SUV), minibus, bus, van, truck. Statistics on the number of different types of vehicles (e.g. heavy vehicles movement through city centres) are useful when making decisions about the infrastructure.

There are a few different approaches to vehicle type recognition described in scientific papers. In the paper [24] two features: size and "linearity" were used. The first one was normalized with using vehicle position information. The second was used to discriminate between trucks and van–trucks or buses. The designed classifier was based on template matching and allowed to integrate cues from different frames. In addition, a shadow elimination procedure used during vehicle segmentation was proposed. The authors reported 93 % recognition accuracy.

In [62] the well-known and reliable scheme: HOG features and SVM classifier was used to divide vehicles into the following categories: motorbikes, small and big cars. The authors reported precision 93.82 % and recall 88 %. The SVM classifier was also used in the work [13]. As

features, simple shape parameters: size, aspect ratio, width and solidity were used. The reported specificity reached 88.7 %.

The recognition can also be based on vehicle shape parameters. For example, in [41] the following were used: width, area, compactness, height to width ratio, major-axis to minor-axis ratio and rectangularity. A two-step kNN classifier was utilized. The reported accuracy was more than 90 %. A very similar approach was applied by the authors of the paper [36]; however, they used a dynamic Bayesian network classifier. In the paper [53] the local binary pattern (LBP) descriptor and linear discriminate analysis (LDA) classifier were used. The authors report an approx. 87 % accuracy.

In the work [3] the vehicle classification was based on Hu geometric invariant moments and a k-NN like approach. Three classes were detected: small, medium, big or unclassified. The method was evaluated on five highway sequences. The authors report 96.96 % accuracy.

Embedded implementations Hardware implementation of vehicle classification was addressed in two scientific papers.

In the work [9] background subtraction for object detection was used. In the post-processing phase the removal of objects with width smaller than N pixels as well as connecting adjacent scan-lines were proposed. The classification was based on vehicle size (kNN classifier). The system was evaluated on a Virtex 4 FPGA device. Real-time image processing for 640×480 pixels was reported.

The study [48] discussed theoretical aspects of the vehicle classification problem implemented in an FPGA device. Methods based on features, model matching and invariants evaluation were compared. In the conclusions the author suggested the third approach, because of its low sensitivity to the camera position and no need for calibration.

Summary The above-described algorithms and their implementations are summarized in Table 3. The following parameters were considered:

- Platform—the used computing platform,
- Features—the used features,
- Classifier—the used classifier,
- Classes—the recognized vehicle classes,
- Evaluation—available information about evaluation, i.e. used dataset and reported accuracy,

The comparison does not include the work [48], because it describes only a concept and not an actually realized vision system.

All described systems involve the following processing stages (Fig. 3):

- obtaining the input sample, i.e. the window (block, ROI) in which the analysed vehicle is present,
- optional scaling the sample to a predetermined size,
- optional adjustment of image parameters: e.g. converting to grayscale, filtration or histogram equalization,
- feature extraction. Depending on the specific approach this could involve: HOG, LBP or some geometrical parameters (for which a high-quality object mask is required),
- classification (SVM, kNN, LDA, etc. classifier).

These solutions allow to assign a given vehicle to one of several fairly coarse classes. Accuracy of the systems differs, but is usually between 80 and 90 %. Once again, it should be noted that each team tested their system on a separate database.

Table 3 Vehicle classification—summary

Work	Platform	Features	Classifier	Classes	Evaluation
[24]	GPP	Size, "linearity"	Template matching	Car, minivan, truck, van truck	Taiwan highways, acc. 93 %
[13]	GPP	Shape based	SVM	Car, van, truck	Specificity 88.7 %
[62]	GPP	HOG	SVM	Car, small, large vehicle, non	Acc. 93 %
[41]	GPP	Shape	Two-step k-NN	2, 3, 4, 6 wheelers (with subclasses)	Sequences from Dhaka Bangladesh and Suwon Korea (1h), acc. 90 %
[36]	GPP	Shape, location and shape of licence plate, vehicle pose	Dynamic Bayesian network	Sedan, bus, micro-bus and unknown	Sequence with 128 vehicles acc. 83.75 %
[53]	GPP	LBP histogram, shape	LDA	Motorbikes, cars, minibuses, trucks and heavy trucks	3400 frames with 76 vehicles, acc. 87.82 %
[3]	GPP	Hu geometric invariant moments	kNN like	Small, medium, big, unknown	Five image sequences (form Taiwan, Italy, France highways), acc. 96.96 %
[9]	FPGA	Size	Linear	Four sizes	Acc. 97–50 % depending on class

Fig. 3 Vehicle type recognition scheme

3.4 Vehicle colour recognition

The topic of vehicle colour recognition was described in several research papers. In the article [65] it was stated that the best car part for obtaining a colour sample is the hood, because it is usually flat and uniform. This area was detected using the shadow cast by the vehicle (a dark region in front of the vehicle). Then, mean brightness was calculated on horizontal lines—to detect uniform and long ones. This information was used to select the sampling area—primarily, the middle part of the hood. In the next step the colour recognition procedure was applied. It was based on HSV colourspace and fuzzy logic approach. The Mamdani Fuzzy Interference System was used. Finally, on a set of 2418 samples 58.60 % accuracy was reported.

In the study [68] also a lot of attention was given to the proper sample selection. Image segmentation was based on k-means clustering algorithm. This allowed to obtain regions with quite similar colour. The object mask was determined using Gaussian Mixture Model (GMM) foreground segmentation. Then, a specially developed procedure (mask-based connected component labelling) and distance transform were used to remove undesired areas such as: tires, windows, shadows and reflections. For colour classification a cascade of two SVM classifiers was used. As feature vectors, histograms in Hue Saturation Value (HSV) colour space aggregated to 16 bins were utilized. The first SVM determined whether the sample was a chromatic or achromatic one. The second recognized the exact colour. For the achromatic: white, grey or black; and for the chromatic: yellow, green, red or blue. The reported accuracy reached 72.3 %.

In the paper [6] a similar recognition scheme was used: HSV histograms as features (only H and S components) and SVM classifier. The authors reported 94.92 % accuracy, however, a description how the samples were obtained was not provided.

In the article [29] seven colours were recognized: black, silver, white, red, yellow, green, and blue. The HSI (Hue, Saturation, Intensity) colour space and 3D histograms were used. The authors analysed the impact of histogram bin number on the final accuracy. The best result, reaching 88.34 % on 700 test images was obtained for 8, 4, 4 bins aggregation for hue, saturation and intensity, respectively.

In a recent paper [26] a more advanced approach to vehicle colour recognition was presented. First a colour correction scheme was introduced. It allowed to compensate the effects of lightning changes. Furthermore, a vehicle window removal procedure was proposed. The features were obtained from RGB and Lab colourspaces. For classification, a tree-based SVM approach was utilized (separate classification of grey and colour vehicles). Seven colours were recognized: red, green, blue, yellow, black, silver and white. The test set consisted of 16,649 samples. An overall accuracy of 93.59 % was reported.

No embedded implementations of these type of vision systems were reported.

Table 4 Vehicle colour—summary

Work	Sample extraction	Colour space	Classifier	Classes	Evaluation
[6]	Well-cropped, frontal images used	HSV (only H and S)	SVM	Black, white, red, yellow, blue	500 test images, acc. 94.92 %
[29]	Well-cropped, frontal images used	HSI (3D histograms)	Shape based	Black, silver, white, red, yellow, green, and blue.	700 test images, acc. 88.34 %
[68]	k-means clustering to obtain areas with uniform colour	HSV (histograms)	Cascade of two SVMs	Black, grey, white, yellow, green and blue	1700 test samples, acc. 72.3 %
[65]	Hood segmentation algorithm	HSV	Mamdani Fuzzy Interference System	Red, brown, orange, yellow, green, blue, purple/violet	sequences from ITS systems, 2418 samples, acc. 58.60 %
[26]	Window removal, colour correction	RGB LAB (advanced features)	Tree + SVM	Red, green, blue, yellow, black, silver and white	16,649 samples, acc. 93.59 %

Summary The above-described algorithms and their implementations are summarized in Table 4. The following parameters were considered:

- sample extraction—the used sample extraction procedure,
- colourspace—the used colourspace,
- classifier—the used classifier,
- classes—the recognized colours,
- evaluation—available information about evaluation, i.e. used dataset and reported accuracy.

The most important issue in the case of vehicle colour recognition is the proper sample extraction procedure. Some of the authors use images with precisely cropped car images. Others propose algorithms to obtain them automatically. The dominant colourspace is HSV. However, it should be noted that for reprogrammable systems it is quite inconvenient due to rather complex conversion from RGB to HSV. Features involve: single colour samples, histograms or more advanced solutions [26]. Also different classifiers are used—from simple sample matching to cascades of SVMs. Typically 5–7 colours are recognized. The accuracy of the systems is different and ranges from 58 to 94 %. Once again, it should be noted that every team used a different database, making a reliable comparison of the methods not possible.

3.5 Summary

The analysis of algorithms used for ITS systems allows to draw some general conclusions. Firstly, the considered subject is very popular and the results can be almost directly applied in industrial applications. The proposed approaches differ from each other. Some are simple while other quite advanced. However, due to reasons pointed out in Sect. 2, in this work, solutions that could be implemented in a pipeline data processing system were preferred. It is interesting that among many cited work, there are not many reports on hardware implementations in reprogrammable devices, i.e. in several papers methods for vehicle detection and counting were proposed and one paper addressed the topic of queue length estimation. Other ITS functionalities were not yet implemented in FPGAs.

Secondly, another characteristic feature is the lack of a common test sequences database. If the *Evaluation* column in Tables 1, 2, 3 and 4 is analysed, it may be noticed that in each of the works different sequences were used. This is a serious obstacle in providing reliable comparison of newly developed algorithm with the "state-of-the-art". The proposal of a reference database appears to be a challenge for the scientific community involved in ITS vision.

Thirdly, the used algorithms have many common elements. In addition, most of them can be implemented in a pipeline data processing system. These features greatly facilitated the hardware–software implementation of exemplary algorithms presented in this paper.

4 The proposed embedded vision system

In the previous section the basic functionalities which should be implemented in an advanced smart camera for ITS were presented: vehicle queue length estimation, vehicle detection and counting, as well as vehicle type and colour recognition. All of these can be implemented in a heterogeneous vision system based on the Xilinx Zynq SoC device. In this paper only exemplary algorithms which implement these functions and their hardware–software realisations are presented. The possibility of implementing more advanced solutions is also pointed out.

The proposed vision system has been implemented on the Zynq SoC device (XC7Z020 CLG484 -1 AP SoC) available at the ZC 702 evaluation board made by Xilinx. The video stream, transmitted in HDMI standard, was supplied to the system via the AES-FMC-DVI-G FMC (FPGA Mezzanine Card) module. During implementation the ISE, EDK and ISim software from Xilinx were used. The general scheme of the proposed system is presented in Fig. 4.

The algorithm has been divided into hardware and software part. During this process the assumptions pointed out in Sect. 2 were used. Therefore, the first choice was a pipeline processing-based hardware implementation. Only for parts of the algorithm, which could not be implemented in the above-described way, a software application on the ARM processor was used.

In the reprogrammable logic the following modules were implemented:

- global modules (*Gauss, RBG YCbCr, LBP, Sobel, CFD, Position*)—Sect. 8,
- vehicle queue length estimation module (*Q_0, Q_X*)—Sect. 5,
- a part of vehicle detection, counting and speed recognition (*VDL_0, VDL_1, VDL_2*)—Sect. 6,
- vehicle type and colour recognition (*VR_0, COLOUR RECOGNITION, TYPE_RECOGNITION*)—Sect. 7.

A part of the vehicle detection module, as well as speed estimation algorithm was implemented in software (ARM processor system). Furthermore, due to the used PetaLinux operating system, it was possible to realise several other functionalities, e.g. communication using UART, ssh, ftp or sftp, a simple database for data logging and a simple website to display detection results and statistics.

Additional elements of the hardware system, presented in Fig. 4, are: *AXI FIFO* buffers used in the data transfer

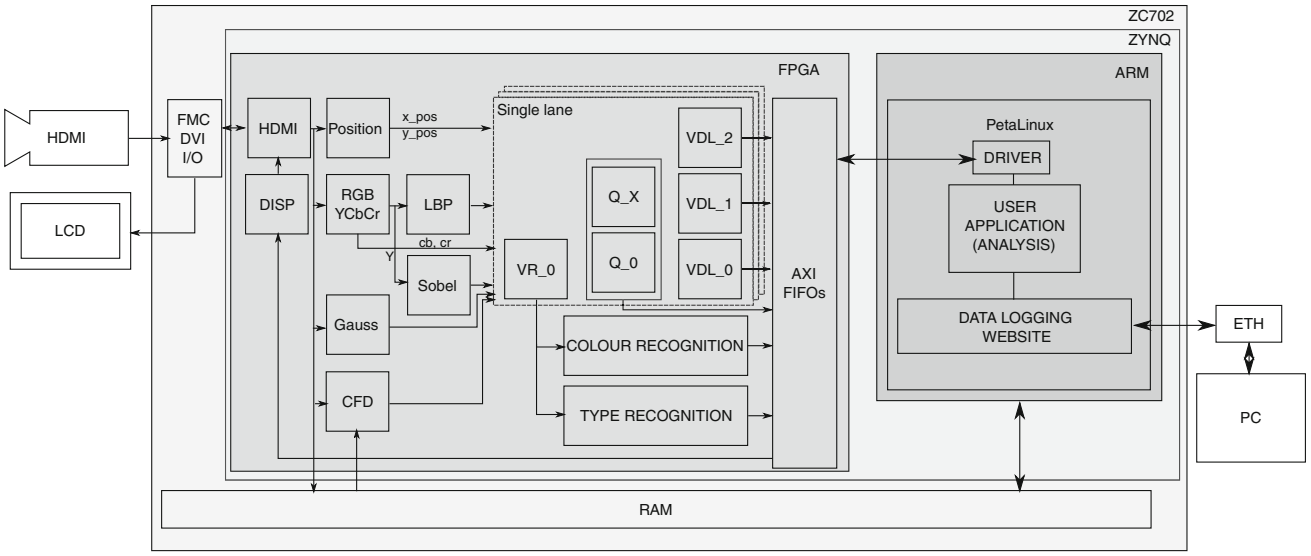

Fig. 4 Scheme of the proposed hardware–software system

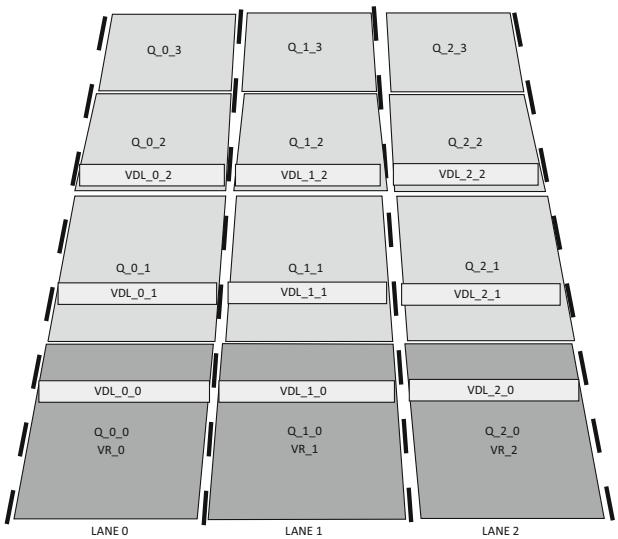

Fig. 5 Scheme of the ROIs location used in the proposed vision system (description in text)

between *FPGA* and the processor system via AXI bus and HDMI—communication with the FMC module, responsible for video stream receiving and displaying (*DISP*).

It is also worth to notice that the *Single lane* module is multiplied three times in the presented version of the system. This is possible due to the parallelism provided by reprogrammable logic devices. On the scheme in Fig. 5 the location of areas (ROIs) used by the system for a three-line intersection is presented. The used symbols (corresponding to Fig. 4):

- Q—queue length estimation module (Sect. 5),
- VDL—virtual detection line module (Sect. 6),
- VR—vehicle type and colour recognition module (Sect. 7).

The numbering convention is as follows. The first digit indicates the lane number and the second the module number (0—closest to the stop line).

In the next sections, first the proposed algorithm is described and then the corresponding hardware–software implementation is presented. In each case, the following procedure was utilized. First, the so-called software reference model was created—C++ implementation with OpenCV library [46], some elements were also prototyped in Matlab software. Then the corresponding hardware modules were designed in Verilog HDL (with the use of IP Cores). They were tested in a simulation tool (Xilinx ISim) and the results were compared with the software model. Finally, the modules were evaluated in hardware. Also the required software application for PetaLinux was developed in C++ and compiled with a cross compiler for the ARM architecture. The integration of the system on the ZC 702 evaluation board with Zynq SoC device from Xilinx is discussed in Sect. 9.

5 Queue length estimation

The thorough scientific papers analysis presented in Sect. 3.1 showed that an effective vehicle queue length estimation algorithm is based on vehicle motion and presence detection. In most cases two methods are used, respectively, consecutive frames subtraction and edge detection and analysis (Summary

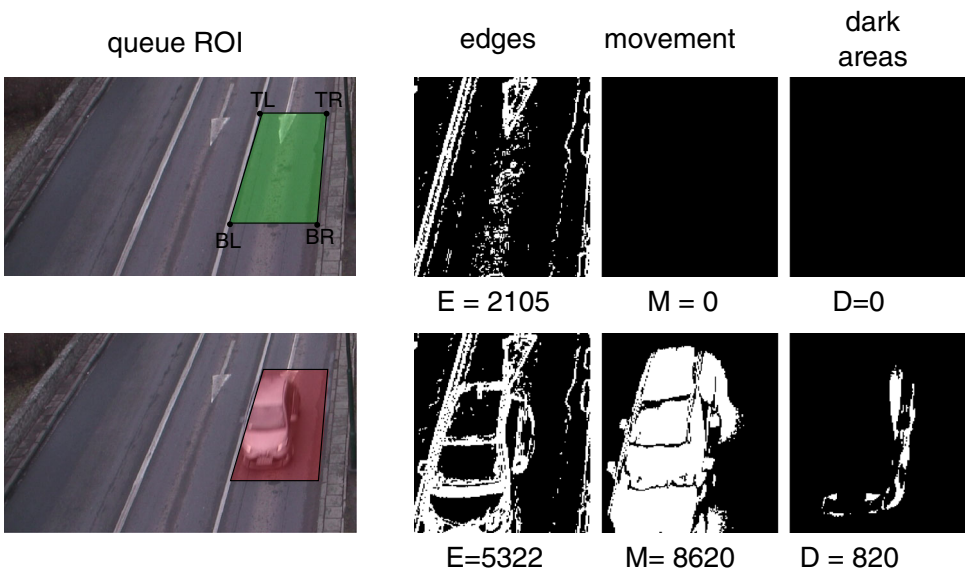

Fig. 6 Vehicle detection example (description in text)

of Sect. 3.1). Therefore, these solutions were also adopted in this work. The proposed module is based on the fairly representative system from the article [55].

5.1 The proposed algorithm

The basic component of the proposed solution is a quadrangular detection block (`queueROI`) defined by four vertices $((x_{TL}, y_{TL}), (x_{TR}, y_{TR}), (x_{BL}, y_{BL}), (x_{BR}, y_{BR}))$: *TL*—top left, *TR*—top right, *BL*—bottom left, *BR*—bottom right). For pixels belonging to the detection area the number of: moving pixels, edge pixels and "dark" pixels is computed.

Moving pixels are determined by thresholding the differential image between two consecutive frames. Edges are detected using the Sobel algorithm. Both modules are described in Sect. 8.

In addition, to determine the number of "dark" pixels (see [55]) it is necessary to calculate the average brightness within the detection area, which is used to compute a binarization threshold (as a certain % of the mean value).

The mentioned information allows to define two flags:

$$\text{blockMov} = \begin{cases} \text{movement} & \text{movSum} > \text{movTh} \\ \text{no movement} & \text{otherwise} \end{cases} \quad (1)$$

where movSum is the sum of moving pixels in a given block, movTh is the constant threshold.

$$\text{blockOcc} = \begin{cases} \text{occupied} & \text{if edgeSum} > \text{edgeTh} \\ & \&\text{blackSum} > \text{blackTh} \\ \text{unoccupied} & \text{otherwise} \end{cases} \quad (2)$$

where edgeSum is the sum of edge pixels in a given block, edgeTh is the fixed threshold, blackSum is the sum of pixels recognized as "dark" within a block, and blackTh is the variable threshold calculated as a certain percentage of the average brightness of pixels within the given block. An extensive justification of this approach can be found in the work [55] (Sections III C and D).

In Fig. 6 an example of the proposed vehicle detection method is presented. In the initial situation (top part) the movSum (*M*) and blackSum (*D*) values are equal to 0. The edgeSum (*E*) is not zero due to presence of edges in the "background"—mainly lane markings. When a car approaches (bottom part) all values are significantly higher and, therefore, the detection is in this case quite straightforward.[4]

Vehicle queue formation in front of traffic lights has a certain specificity (i.e. is subjected to certain rules). When the drivers behave typically (do not leave big gaps between vehicles) the process is sequential. First, the place closest to the stop line is occupied. Then, the cars stop at subsequent places. It is assumed that the queue disappears when the first car moves (i.e. green light turns on).

This observation can be used to optimize the computing resources, both in software and hardware implementations. Rather than implementing a number of detection modules on a single lane (usually minimum four) only two are required. One, fixed for the continuous monitoring of the head of the queue and a second, movable, for queue tail detection. An example is presented in Fig. 7.

In the initial situation only the first module (*row* 0) is active. When a stopped vehicle is detected (*no movement* and *occupied* conditions fulfilled), the second module (*row* 1) is activated and the queue length counter *Q* is

[4] The movement mask may look like the result of foreground object detection, but the clear vehicle silhouette is the consequence of its rather high speed.

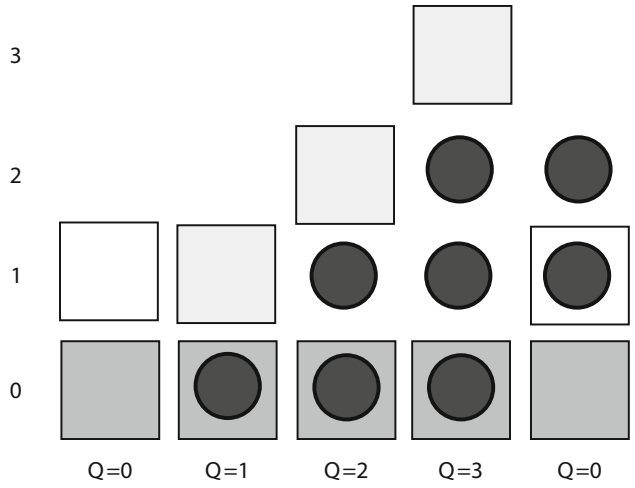

Fig. 7 Vehicle queue formation example for a single lane. *Vertical axis* vehicle detection ROIs. *Horizontal axis* different time moments. *Q* estimated queue length. *Dark circle symbol* vehicle

incremented. At the same time the first module constantly monitors the status of the first vehicle in the queue (to detect when it will start to move). Once another stopped vehicle is detected, the module (light grey) is "being moved" to the next location. When motion is detected at the beginning of the queue the counter is reset to zero.

The algorithm was evaluated in the C++ software model on several sequences. It allowed to reliably estimate the queue length. However, the thresholds had to be selected carefully. This is related to two issues. First, for some locations at the intersection edges are also present in background (e.g. lane markings). Therefore, the threshold value has to be higher. Second, due to perspective the ROIs have different sizes. This also should be considered when selecting the threshold.

5.2 Hardware implementation

A scheme of the hardware module which realises the functionality of a single detection block is presented in Fig. 8.

The basic element is the logic which allows to determine whether the currently processed pixel [with coordinates (x, y)] lies inside a given block (`queueROI`). In the scheme it is described as `xcos(a)+ysin(a)-d`. The line equation in the normal form is used to avoid problems with representing vertical lines. At the configuration stage parameters: $\cos(a)$, $\sin(a)$, d are computed from the ROI vertices coordinates. Additionally, the information about correspondence between the line and the block area is stored (determines which inequality should be used: greater or lesser). These parameters are marked as `line_params`. The current position is given by coordinates `pos_x`, `pos_y` (obtained in the position computing module—ref. Sect. 8). The information about the pixels position to the given ROI is denoted by the `inside_ROI` flag.

The second component is a set of three adders (`SUM_`) connected to thresholding modules (`TH`). Their task is to count the number of pixels classified as moving (`mov`), edge (`edge`) and black (`black`).

The remaining logic is a module which allows to compute the mean brightness in the ROI. It consists of an accumulate unit `SUM_GREY` and a divider `DIV(ROI)`. The divider was implemented as iterative. This is feasible,

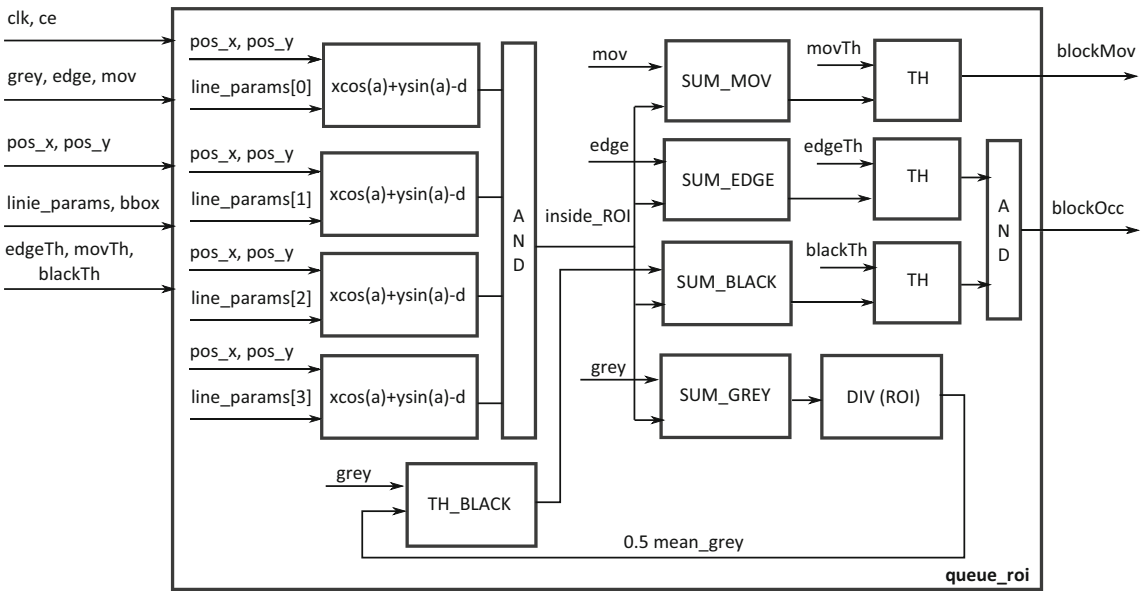

Fig. 8 Scheme of the hardware module, which realises the functionality of a single detection block `queueROI` (detailed description in text)

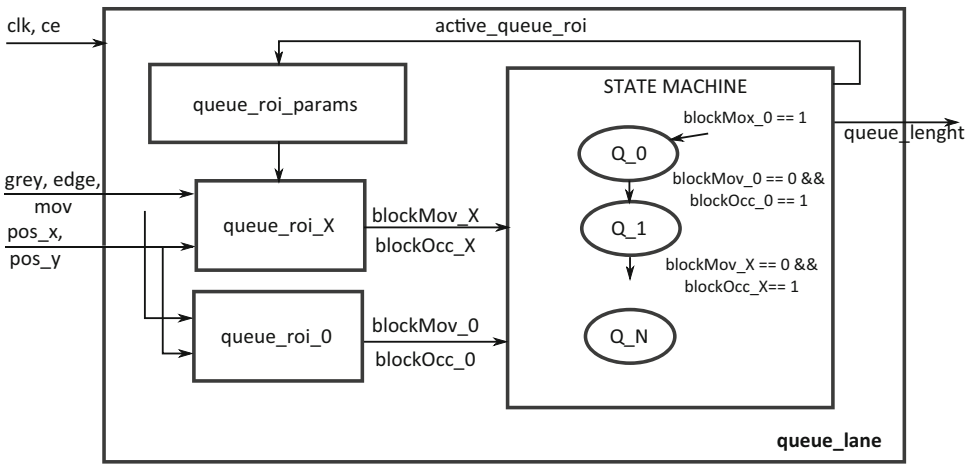

Fig. 9 Vehicle queue length estimation module for a single lane (detailed description in text)

Table 5 Vehicle queue length estimation module for a single lane resource usage

Resource	Used	Available	Percentage (%)
FF	393	106,400	0
LUT	1087	53,200	2
DSP 48	20	220	10

as the threshold obtained in iteration i is used in iteration $i+1$—due to pipeline image processing concept. The threshold value is used in the module `TH_BLACK`. Outputs of the module are binary signals: blockMov and blockOcc.

The scheme of the proposed vehicle queue length estimation module for a single lane is presented in Fig. 9.

It consists of two modules `queue_roi`. The first (postfix `_0`) corresponds to the detection ROI directly in front of the traffic lights. The second (postfix `_X`) "tracks" the queue tail. It is controlled by parameters stored in the `queue_roi_params` block, which are selected by the signal `active_queue_roi`. Situation analysis on a lane is realized using a state machine. Particular states correspond to the number of blocks with detected stopped vehicles. The transition to the next state is possible when `blockMov == 0` and `blockOcc == 1`. The machine is reset when `blockMov_0 == 1` – movement in the first block. Output of the module is the queue length estimation expressed as the number of occupied ROIs `queue_length`. The resource usage is summarized in Table 5.[5] The DSP module usage is quite high, due to `inside_ROI` flag computation. However, in case of insufficient DSP resources, fabric-based multiplier could also be considered. The estimated maximum frequency equals 276 MHz after synthesis and 155 MHz after place and route.

[5] All the presented resource usages, if not stated explicitly, are estimations obtained by the ISE software after synthesis.

Finally, it should also be noted that in a similar way it is possible to implement solutions based on the Harris corner detection [1] or features detector used in the paper [51]. Only the Harr features and AdaBoost algorithm [72] are quite difficult to implement in a pipeline vision system because of the large logic resource requirement.

6 Vehicle detection and counting

The presented vehicle detection algorithm was initially proposed in [33] as the first element of a Zynq SoC-based smart camera for ITS systems. Several factors were taken into account while designing the algorithm: accuracy, computational complexity, the possibility to divide the computations between hardware and software, resistance to various lightning conditions (time of the day, shadows) and camera jitter, as well as the ability to work in conditions occurring at a crowded intersection. The last factor requires an extended comment. Some approaches described in the literature [41, 52] were designed and tested on sequences recorded by cameras mounted over a road, where the vehicle movement is usually smooth (e.g. highway). In such cases methods based on background subtraction or optical flow can obtain quite good results, because the road surface is visible for the most of the time and the vehicle speeds are quite high.

In this work it was assumed that the system should be able to operate at a typical intersection, where many cars stop at a red light. This condition, as well as the characteristics described above, led to the conclusion that background modelling or optical flow methods are not an adequate solution for the designed application. It was, therefore, decided to use an approach based on virtual detection lines (VDL) and time-spatial images (TSI).

To reduce the impact of external lighting conditions and camera jitter, all operations were performed on two

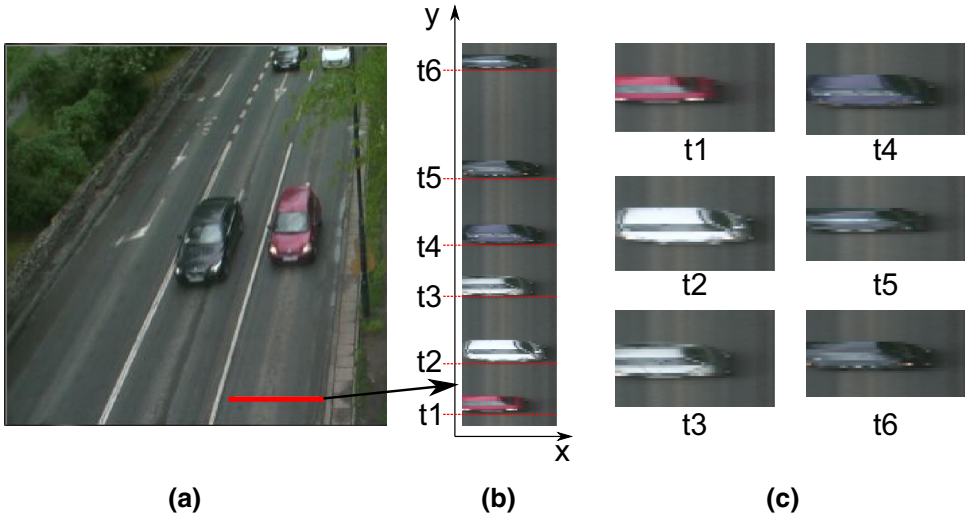

Fig. 10 Scheme of the proposed vehicle detection and counting system. **a** VDL located on the road, **b** TSI, **c** obtained patches [33]

consecutive frames. The concept is based on detecting the presence of vehicles by analysing only the local neighbourhood of a VDL—this issue is described in details in Sect. 6.1. As a result, small images (patches) containing vehicles are obtained. In the second stage, the patches are analysed to eliminate erroneous (caused by shadow or other disturbances) or multiple detections—a detailed description is provided in Sect. 6.2. The idea is schematically shown in Fig. 10. The evaluation of the proposed solution is presented in Sect. 6.3. The designed method was inspired by the approaches described in the literature, particularly in the works [41, 73], however, it has also new elements specific for a hardware–software system.

6.1 Vehicle presence detection

The vehicle presence detection is based on identifying similarities between two successive (in terms of time) VDL neighbourhoods of size 3 × VDL width. The scheme of the proposed solution is presented in Fig. 11. I_N and I_{N-1} stand for the current and previous VDL context. In the first step a horizontal Sobel gradient (*SOBEL X*) and 3 × 3 LBP transform (*LBP 3x3*) are computed.

The basic LBP transform for a given pixel is formed by thresholding the 3 × 3 neighbourhood with the centre pixel value as the threshold. It was first introduced by Ojala [44] and used to describe local texture features. Let i_c be the intensity of the centre pixel and $i_n(n = 0, 1, \ldots 7)$ pixel intensities from the neighbourhood. Then the LBP is given by:

$$\mathrm{LBP} = \sum_{n=0}^{P-1} s(i_n - i_c) \cdot 2^n \tag{3}$$

where P is the number of sample points and

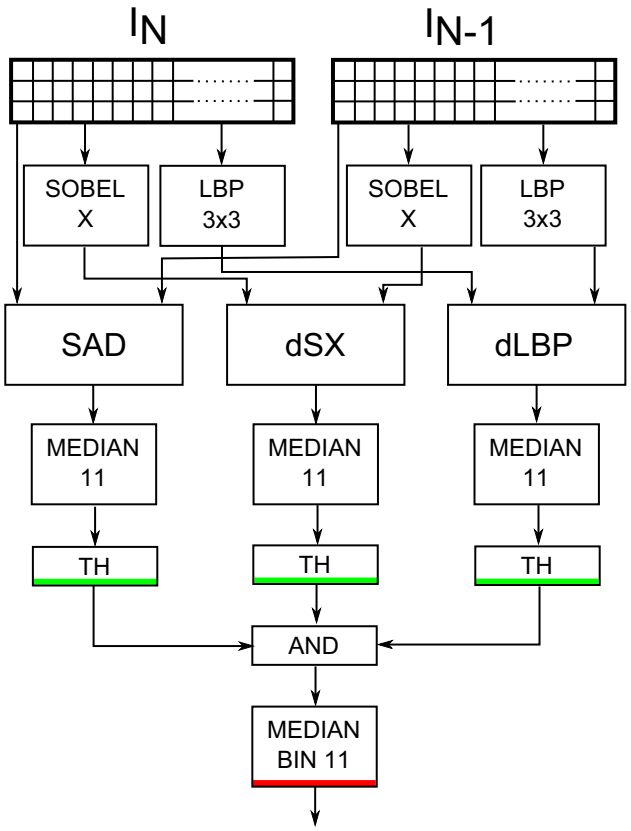

Fig. 11 Scheme of the proposed vehicle presence detection algorithm [33]

$$s(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{4}$$

In this case the LBP could be interpreted as an 8-bit integer.

As similarity measures are used:

– SAD:

$$\mathrm{SAD} = \sum_{i=0}^{3} \sum_{j=0}^{L} \sum_{k=0}^{3} |I_N^k(i,j) - I_{N-1}^k(i,j)| \tag{5}$$

where i is the line index in the VDL neighbourhood (vertical), L is the width of the detection line, I_N is the Nth image frame in RGB colour space, k is the particular colour component {R, G, B}.

– absolute value of differences between edge images (computed using horizontal Sobel gradient):

$$\mathrm{dSX} = \sum_{i=0}^{3} \sum_{j=0}^{L} \sum_{k=0}^{3} |\mathrm{SX}_N^k(i,j) - \mathrm{SX}_{N-1}^k(i,j)| \tag{6}$$

where SX_N is the horizontal Sobel gradient for Nth video frame.

– Hamming distance computed for binary 3×3 LBP transform result obtained for consecutive frames:

$$\mathrm{dLBP} = \sum_{i=0}^{3} \sum_{j=0}^{L} \sum_{q=0}^{8} \mathrm{LBP}_N(i,j,q) XOR \mathrm{LBP}_{N-1}(i,j,q) \tag{7}$$

where LBP_N is the LBP transform result for Nth frame, q the index in an 8-bit vector (result of the LBP transform for a 3×3 window).

The resulting similarity measures: SAD, dSX and dLBP are subjected to one-dimensional median filtering (window size 11 samples) (module *MEDIAN 11*) and thresholded (module *TH*). The binary results are combined with the *AND* operator and subjected to another binary one-dimensional median filtering (window size 11 samples) (module *MEDIAN BIN 11*). Finally, a binary flag containing information about vehicle presence is obtained.

Sample TSI images are presented in Fig. 12. The binarization results of the three similarity measures are visualized as light grey (1)/white (0) columns on the right side of the TSI. The dark grey colour indicates fragments without car detection (i.e. carriageway). The delay introduced by the used median filtering is compensated later by extracting shifted fragments (on the presented images a 10–15 pixel "down" shift is necessary).

It is worth noting that in the TSI in Fig. 12b (at the bottom) a part of the dark car is erroneously detected as a carriageway. On the other hand, in TSI in Fig. 12c (at the top) the distance between two cars turned out to be too small, and they could not be separated properly at this stage of the algorithm. These two issues will be addressed in the following subsection.

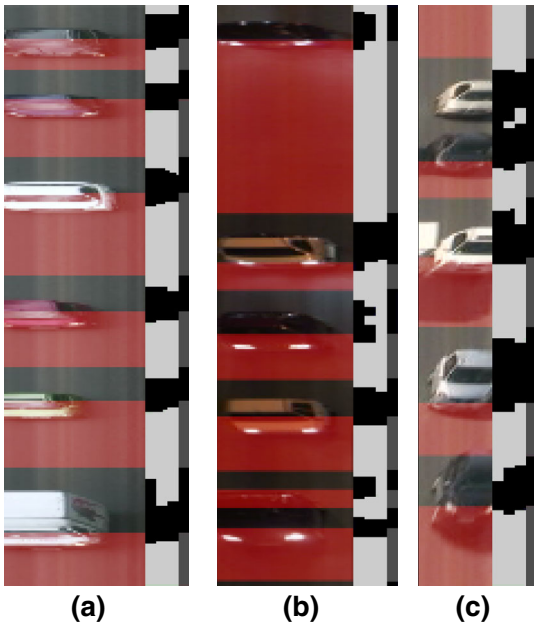

Fig. 12 Sample results of the proposed vehicle presence detection method [33]

6.2 Patch analysis

Based on the detection results for a number of test sequences it has been found that in most cases the method described above can correctly extract individual vehicles. However, also situations when a patch contains more than one car or no car at all (in case of shadows) have been noticed. Furthermore, a slight difference in brightness or colour between the vehicle and the road, as well as obstructions, causes serious classification problems. The latter issue can be partially eliminated by appropriate positioning of the camera—directly over the road and at a fairly large angle. Sample patches, with erroneous detections, are presented in Fig. 13.

Algorithms that allow to determine the number of vehicles in a given patch were proposed to improve the overall accuracy of the system. The solution can handle cases presented in Fig. 13a–d. The other two situations should be considered as difficult ones and their correct classification requires further research.

Due to a completely different specificity, separate analysis procedures for day and night-time were developed.

Analysis during day-time The proposed method consists of the following steps:

– low-pass filtering (Gaussian)—distortion elimination and image smoothing.

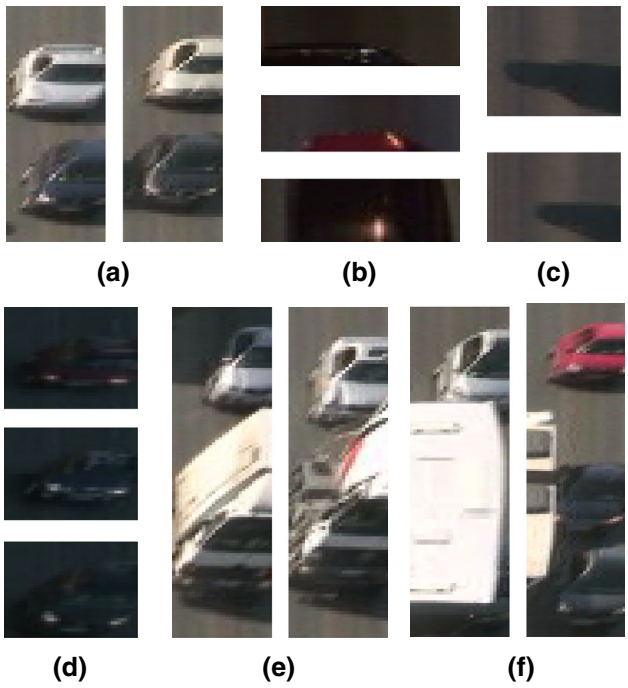

Fig. 13 Patches requiring further analysis: **a** more than one vehicle in the patch, **b** incorrectly detected patch (during night-time), **c** detected shadow (cast by a car on a neighbouring line), **d** very low contrast between a vehicle and the background, **e** a vehicle partially obstructed by a larger one, **f** a vehicle on adjacent line causes distortions (in the left image it completely obscures the adjacent lane) [33]

- histogram stretching—a procedure similar to the available in Matlab software was used. There, 1 % of the brightest and darkest pixels is saturated, respectively, at the values 0 and 255.
- another low-pass filtering (Gaussian)—the histogram stretching operation emphasizes noise present in the image and it should be removed prior further image analysis.
- calculation of the horizontal Sobel gradient—information about horizontal edges is used to determine the presence of an object and to eliminate shadows. The gradient is calculated separately for each RGB colour component, then the result is thresholded and combined with an OR operator.
- shadow detection—to detect situations similar to the shown in Fig. 13c a simple method was proposed that analyses edges and shadow areas (as a shadow are regarded pixels with brightness below a certain threshold). The approach is illustrated in Fig. 14. In the first step areas with horizontal edges and shadow pixels are determined. Then the bounding box around the edge area is computed. This allows, in most cases, to divide the patch into two parts: one containing the object (edges) and second with the background (no edges). For these parts greyscale histograms aggregated to 64 values are calculated. Then, the histograms are subtracted from each other, and, in addition, the histogram part corresponding to the shadow areas (below a given threshold) is removed. Finally, values present in the obtained histogram are summed and then normalized by a number equal to the size of the detection window. For the case presented in Fig. 14 the resulting value is quite low (0.0827). If a vehicle is present in the patch, then the value corresponding to it brightens and will be visible on the resulting histogram, thus the coefficient will be high. In addition, two factors are calculated: the ratio of the number of pixels marked as edges to the number of pixels marked as a the shadow, and the ratio of the number of pixels marked as edges and the shadow to the bounding box area around the edges. For the patch presented in Fig. 14 these values are, respectively, 0.74 and 1.29. It was assumed that for shaded areas the first coefficient should be smaller than 2, i.e. the edge and shadow areas should be similar size (for vehicles the edge area is usually greater than the area of the shadow) and the second coefficient should be greater than 0.6, i.e. the bounding box area is mostly "filled" with points belonging to edges and shadows. To classify a given patch as containing the shadow all three conditions should be fulfilled. It is worth noting that the above conditions are also met by very dark vehicles, especially in low-light conditions (cloudy day, rain). One possible solution could be the implementation of a deep shadow detection procedure—for example, by analysing the shadow cast by a permanent element of the scene.
- analysis of patches with multiple vehicles—in the proposed method it is assumed that the carriageway is visible between vehicles (see Fig. 12a). The approach is based on region growing segmentation performed on horizontal lines, separately for the left and right side, starting from the patch boarder. The threshold required to determine if the current pixel is similar to the previous ones is determined adaptively as 0.25 $\times$ current pixel value and 0.75 $\times$ previous threshold value. The segmentation is performed in RGB colour space. The resulting masks (for the left and right part) are subject to morphological dilation and then combined by the AND operator. In this way, portions of the carriageway are detected. In the final step the number of separate vehicles is determined. It should be noted that this analysis is performed only for patches with a height greater than a specified threshold. The method is illustrated in Fig. 15. During preliminary research some other approaches to this issue were evaluated. First, the possibility of using information about colour to determine the number of vehicles present on a patch

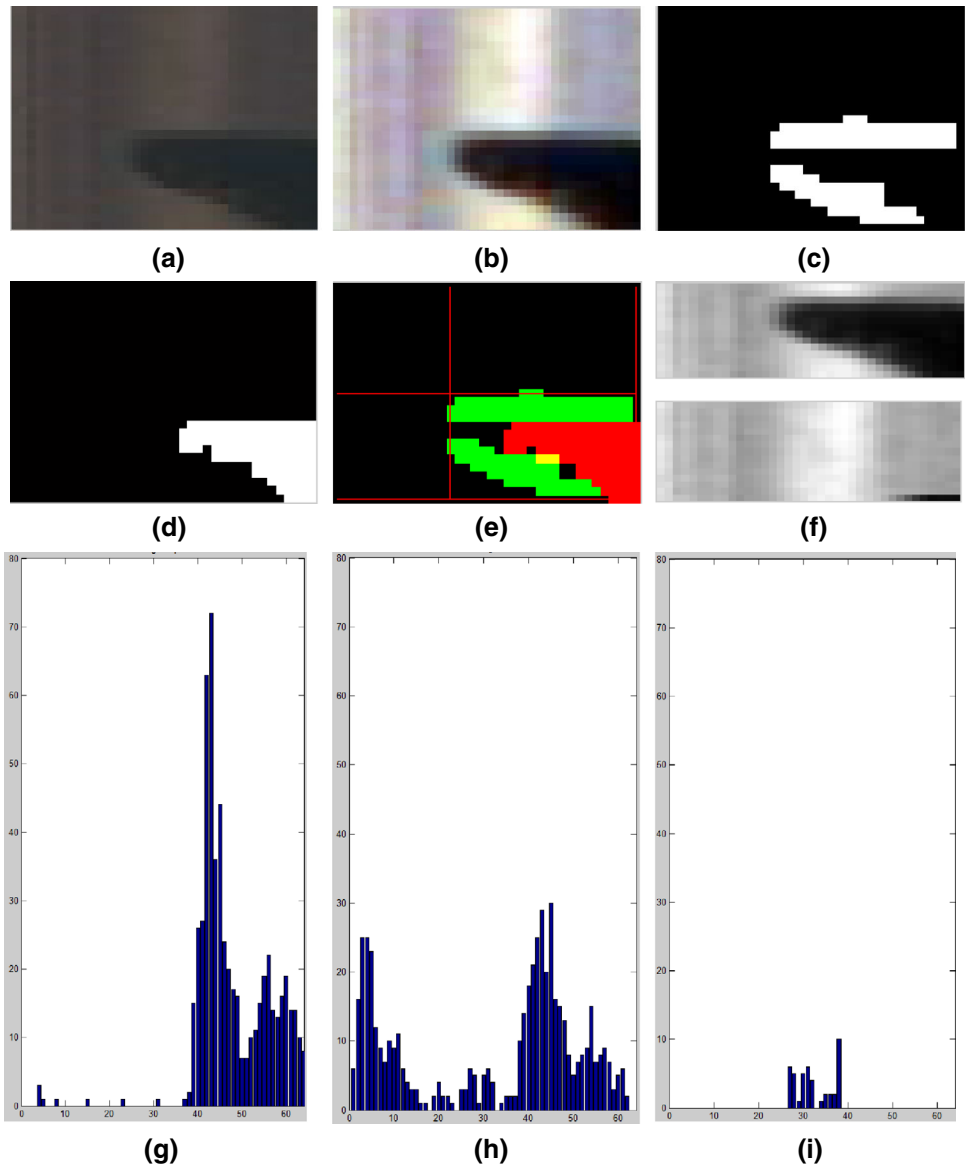

Fig. 14 Sample shadow detection: **a** input image, **b** image after histogram stretching, **c** horizontal edges, **d** shadow areas, **e** combination of images **c** and **d** with bounding box around the edges, **f** part with the edge area (*top*) and without edges (*bottom*), **g** histogram for the non-edge area, **h** histogram for the edge area, **i** difference between histograms **g** and **h** with removed values corresponding to shadow areas [33]

was tested. Experiments in HSV, YCbCr and CIE Lab colour spaces were performed. Unfortunately, analysis of the obtained histograms revealed that this approach can work only for vehicles with a very clear difference in colours (e.g. red and green). In other cases it was impossible, even "manually", to point out at the histogram the maximum corresponding to a given vehicle. The probable cause is the fairly inhomogeneous lighting, including reflections and shadows. Second, local background modelling for a VDL was evaluated. The information could be very useful in vehicle segmentation on a given patch. However, problems similar to those occurring when modelling the background for the whole scene were noted—mainly with distortions caused by shadows. The development of methods able to handle the difficult cases—Fig. 12e and f should be part of future research.

– estimation of the final number of vehicles—the analysis described above is supplemented by counting the overall number of horizontal edges present in the patch. If the number is less than a given threshold, the patch is regarded as a false detection (i.e. without a vehicle).

Analysis during night-time The basis of the patch analysis method during nigh-time is the detection of vehicle headlights. First, binarization of the greyscale image with a quite high threshold (200) is performed, followed by a single-pass connected component labelling. The area and centroids of detected object are computed. Finally, the presence of two objects with similar vertical and different horizontal coordinates is determined.

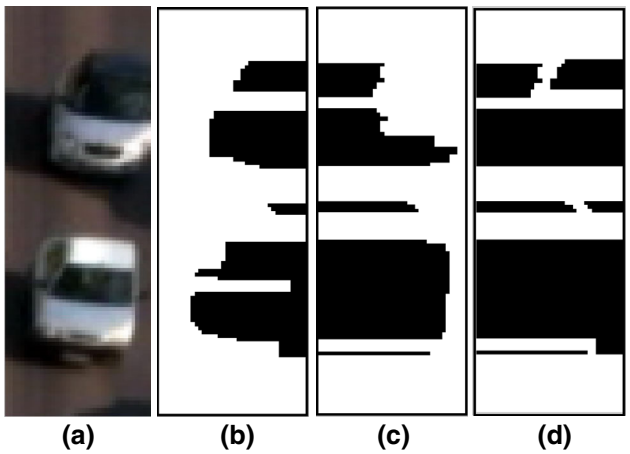

Fig. 15 Sample multiple vehicles detection. **a** Input image, **b** and **c** region growing segmentation results, the procedure started for the left and right edge of the patch (additional morphological dilation was performed), **d** combination of images **b** and **c**—AND operator [33]

6.3 Algorithm evaluation

The algorithm was implemented in C++ language using the OpenCV library as a part of the designed software model and evaluated on a number of test sequences registered by a camera located above a busy intersection on one of the main streets in Krakow. Sequences were recorded under different conditions: sunny day, cloudy day, rainy day and night-time. In total over 53,000 frames were analysed (about 30 min). It is also worth mentioning that all the tests were performed on the same set of parameters, i.e. they were not fine-tuned to particular sequences or even weather conditions.

During evaluation the total number of detections (actual and returned by the algorithm), as well as results of particular patches analysis, were counted. In the second case, the typical binary classification measures were used:

- TP—true positive—correctly detected vehicle,
- TN—true negative—correctly detected road or distortion,
- FP—false positive—road detected as vehicle,
- FN—false negative—vehicle detected as road.

The obtained results are presented in Table 6. The mean accuracy of the proposed system was 96 %.

The main cause for false positives was vehicles stopped at the VDL, which were sometimes counted more than one time. Almost all false negatives were caused by black cars misclassified as shadow. Therefore, the day-time procedure certainly requires some refinement. In the case of sequences registered at night-time, all the missed detection were the result of low brightness of the headlights.

It is worth mentioning that the method allowed to eliminate many potential false detection, which is indicated by the quite high number of true negatives.

6.4 Vehicle speed estimation

In the proposed system, due to the fine grain parallelism provided by reconfigurable resources, it is possible to easily implement more than one VDL on a single lane. This has two advantages. First, it increases the accuracy and reliability of the system, as the vehicle counter is based on multiple semi-independent detectors.

Second, a rough speed estimation can be implemented. It is based on vehicle detection time (binary information from a given VDL)[6] and number of frames per second (fps). This allows to convert the number of frames between vehicle detection on two consecutive VDLs to time and after calibration to speed (knowing the actual distance between the VDLs in meters). In the presented solution three detection lines are used. They are placed every 3 m. In case of a vehicle detection, the frame number is stored. After detecting the same vehicle on all lines the mean speed is estimated using the above-mentioned relationships. The module was implemented as a part of the software application (ARM processor system). The conducted tests showed that the method allows to roughly estimate the vehicles speed. However, correct results are obtained only when the movement is smooth, i.e. there is no traffic congestion.

6.5 Hardware implementation

All operations that are common to the VDL modules (colour space conversion from RGB to greyscale (Y component in YCbCr), Sobel edge detection, Gaussian filtering, LBP transform) were performed for the entire video frame—Sect. 8. Preprocessed data are fetched into a single VDL module, whose scheme is analogous to those presented in Fig. 11. During development of the module, it turned out that implementing Bitonic Merge Sorting based median filtering is very resource consuming. Therefore, contrary to our previous work [33], this operation was moved from hardware to the software part.

All modules were implemented in Verilog hardware description language. The hardware modules results were compared with those obtained from the software application (C++) in the ISim simulation tool. Resource usage for a single VDL module is summarized in Table 7. The estimated maximum frequency equals 549 MHz after synthesis and 257 MHz after place and route. A single VDL transmits RGB pixels values (after Gaussian low-pass filtering), as well as computed similarity measures and detection results to the software part.

[6] In the solution the time measure corresponds to the frame number.

Table 6 Evaluation of results of the proposed vehicle detection and counting algorithm

Sequence	Cars—actual	Cars—detected	No. of patches	TP	TN	FP	FN
Sunny	82	83	96	76	2	11	7
Cloudy	179	166	210	163	28	3	16
Rainy	123	116	151	116	28	0	7
Night-time	93	88	116	88	23	0	5
Overall	627	601	765	587	114	22	42

Table 7 Virtual detection line resource usage

Resource	Used	Available	Percentage (%)
FF	570	106,400	0
LUT	396	53,200	0
DSP 48	3	220	1
BRAM	2	140	1

6.6 Software part

The analysis algorithm, described in Sect. 6.2, was implemented as a user application. It was divided into three parts:

- patch creation using data obtained from the hardware part, patch saving,
- analysis in day-time conditions,
- analysis in nigh-time conditions.

The user application displays informations about detections, as well as other diagnostic data on a console. Additionally, it is possible to connect to the system using secure file transfer protocol (SFTP) and download the obtained vehicle patches.

7 Vehicle type and colour classification

At the concept phase of the vehicle classification module design it was assumed that it should work only in case of smooth vehicle motion. As it was stated earlier, the correct analysis in a traffic congestion situation is quite difficult (e.g. problem with separating particular vehicles).

Vehicle type and colour recognition is based on the analysis of an image patch (ROI) of size 160×224 pixels. In the test setup this parameter allows to capture images of virtually every passenger car. An example is presented in Fig. 3. For larger vehicles: minibuses, buses and lorries only the front part is captured. However, this has no significant influence on the recognition, as the middle and back parts of those vehicles are usually not very informative.

An alternative solution would require the implementation of a reliable vehicle end detection algorithm. Then the ROI should be cut out and rescaled to a defined size. Due to high variability of sizes, a pipeline hardware implementation of such a solution seems to be quite cumbersome. Also the alternative approach of multi-scale image processing is resource consuming.

Using the above-described mechanism two approaches are possible. In the first, computations are performed all the time, i.e. for every frame irrespective of the ROI content. In the second, the module operates only when a vehicle is detected. It should be noted that in the first variant the information about vehicle presence is also essential, because only the "valid" detector responses should be analysed.

The proposed vehicle presence detection mechanism is described in Sect. 7.1, vehicle type and colour recognition, respectively, in Sects. 7.2 and 7.3.

7.1 Vehicle presence detection

In the initial research, the vehicle presence detection was based on the prior described VDL approach (Sect. 6). Unfortunately, it turned out that due to the quite significant delay imposed by the detector (resulting from the used median filtering) and the dependence of the detection moment on vehicle speed, this mechanism could not be used. The obtained vehicle location inside the ROI was quite variable, which is undesirable in standard recognition algorithms: HOG + SVM, LBP + SVM.

Therefore, another approach was used. Since it was assumed that only moving vehicles will be analysed, it was decided to use a movement based presence detection procedure. The idea is shown in Fig. 16. There, two separated zones marked as $A1$ and $A2$, each 8 pixel high, are visible. Within them, moving pixels obtained by consecutive frame differencing (Fig. 16b) are summed and stored as C_{A1} and C_{A2}. A vehicle is detected when the C_{A1} value is greater than C_{A2} and C_{A1} exceeds a certain threshold. To avoid multiple detections of the same vehicle, also the vehicle end must be located. This is done using "symmetrical" conditions, i.e. C_{A2} greater than C_{A1} (C_{A2} greater than a certain

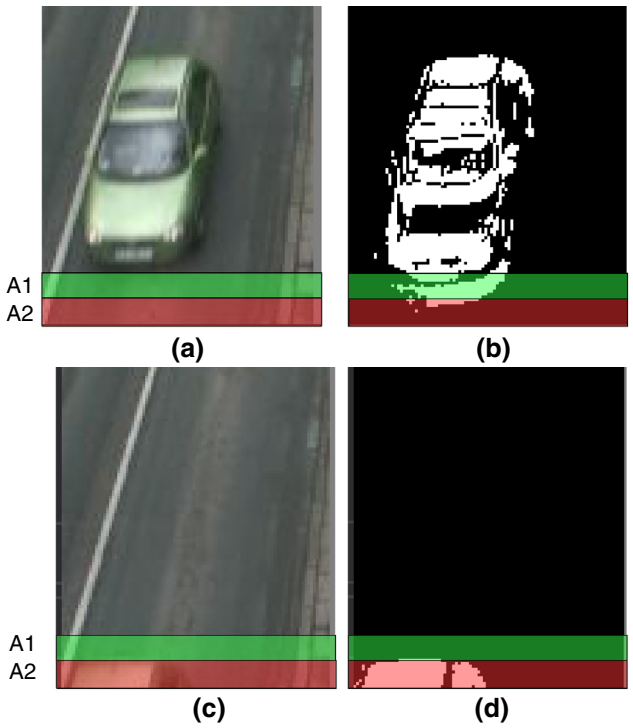

Fig. 16 Vehicle presence detection example: **a** and **b** detection of the vehicle front: $C_{A1} = 130$, $C_{A2} = 0$, **c** and **d** detection of the vehicle end $C_{A1} = 0$, $C_{A2} = 359$

threshold, C_{A1} smaller than a certain threshold) . In addition, no moving vehicle is present if $C_{A2} == 0$ and $C_{A1} == 0$.

The hardware implementation of the described module is quite straightforward. It is based on two counters C_{A1} and C_{A2} and a mechanism to determine whether the currently considered pixel is located inside the area $A1$ or $A2$. On this basis, using the above-defined conditions, the binary value `vehcileDetection` is computed.

7.2 Vehicle type classification

In the described vision system vehicle classification is based on the scheme LBP and SVM . It is a compromise between ease of implementation usage and efficiency of the classification. However, it is worth stating that both used features and the classifier could be replaced depending on the application requirement. For example, HOG could be used. Hardware realisation of such a vision system is presented in one of our previous works [30].

In the sample design, three vehicle classes are recognized: estate, sedan and minibus (they were present at the used test sequences).

Local binary patterns The LBP descriptor can be used to describe local texture features (compare Sect. 6). However, it also turned out to be useful in face detection and recognition, pedestrian detection and other pattern recognition tasks. The main advantages of the LBP are low computational complexity and invariance to local illumination changes.

The basic LBP feature, described by Eq. (3) can take 256 values. In the literature also other variants were proposed. In [45] the concept of *uniform* LBP was proposed. Only patterns with up to two transitions between 0 and 1 are considered. In this way ULBP describes simple texture features: edges, curves, etc. For a 3×3 context in can take 59 values.

In the work [43] *non-redundant* LBP concept was introduced. In this case LBP patterns and its complements are considered as the same (e.g. 10010011 and 01101100). This, combined with ULBP for NRULBP gives maximal 30 different values for a 3×3 context.

Generating basic LBP patterns in uniform image patches could lead to unreliable results as the absolute differences between the centre pixel and its neighbourhood are relatively small. One possible solution is to modify the threshold by:

- adding a bias to the threshold,
- using as the threshold the mean of the 3×3 neighbourhood,
- using as the threshold the median of the 3×3 neighbourhood.

These three approaches were evaluated in the proposed system.

Object recognition with LBP In the proposed system, vehicle recognition takes place in predetermined locations. This is a significant simplification with respect to a typical sliding widow mechanism, where the entire frame is searched (also in multiple scales). The detection window, separate for each lane, has a size of 160×224 pixels and is located at the beginning of the lane (Fig. 10).

In the first step, for each pixel inside the window an LBP descriptor is computed. In the next step, the window is divided into K separated blocks (e.g. 4×4, 8×8 or 16×16 pixels) in which a histogram is computed:

$$h_{k(i)} = \sum_{x \in k} T(\text{LBP} == i), \quad i = 0, 1 \ldots B - 1 \tag{8}$$

where h_k is the histogram of the block k, $T(z) = 1$ when z is true, else $T(z) = 0$, B the number of histogram bins (depending on the used LBP variant, respectively, 30, 59 or 256).

The histogram values are normalised to the [0–1] range by dividing the bins by the sum of all elements.

$$hn_{k(i)} = \frac{h_{k(i)}}{m \cdot n}, \quad i = 0, 1 \ldots B - 1 \tag{9}$$

where $m \times n$ is the block size. The histogram from a given block is formed into a vector:

$$V_k = \left| hn_{k(0)}\ hn_{k(1)}\ \ldots\ hn(k)_{(B-1)} \right| \tag{10}$$

The vectors from all blocks within a window form the feature vector:

$$F = |V_1\ V_2\ \ldots\ V_K| \tag{11}$$

The obtained feature vector is an input for the classifier. In this work, the SVM classifier is used, which proved to provide good detection results in our previous research [32] and also is widely used in classification applications. The SVM theory was presented in the work of [58]. The SVM in this form is based on dividing the feature set by a hyperplane and therefore able to do a binary classification (negative/positive). It is given by the following equation:

$$r = \text{sign}(w \cdot x + b) \tag{12}$$

where $w \cdot x = \sum_i w_i x_i$ and x_i is the ith feature in the feature vector and w_i is its corresponding weight, b is a bias term. The values of w_i and b are obtained during the training process in which training images are divided into two groups, one containing only images with objects of a class that should be positively classified (e.g. dogs), the other objects that do not belong to this class (cats, giraffes, snakes, etc.).

The above-described SVM is linear and, therefore, suitable only for problems where the positive and negative samples can be separated by a hyperplane. If this is not the case, the so-called kernel trick should be used. It allows to transform the feature space to a higher dimensionality, with a non-linear transformation. The frequently used kernels involve: polynomial, Gaussian radial basis functions and hyperbolic tangent.

Once trained, thanks to the large amount of training data, the SVM classifier becomes tolerant to shape differences and viewpoint changes. Unfortunately, it is not very tolerant to scale differences. However, in the proposed application the vehicle sizes are rather constant due to fixed setup.

The SVM is a binary classifier, i.e. able to separate only two classes. In the described system at least three object classes should be recognized. A common approach to this problem is the use of multiple SVM. Each of them separates one class from the others. In this case:

- SVM_1—(estate) vs. (sedan, minibus),
- SVM_2—(sedan) vs. (estate, minibus),
- SVM_3—(minibus) vs. (sedan, estate).

During classification the r value (Eq. (12)) for all SVMs is computed. As the final recognition the class corresponding to the maximum r value is chosen (often r should exceed a threshold to avoid uncertain classifications).

The proposed algorithm In the first stage of designing the algorithm an image database of vehicles was collected. Initial experiments showed that it should be possible to recognize three vehicle types: estate, sedan (a characteristic edge in the back) and minibus (characteristic front). Image samples are presented in Fig. 17.

In total in the experiments 102 samples of class estate, 39 sedans and 25 minibus were used. They were divided randomly into training and test set (approximately 50 %).

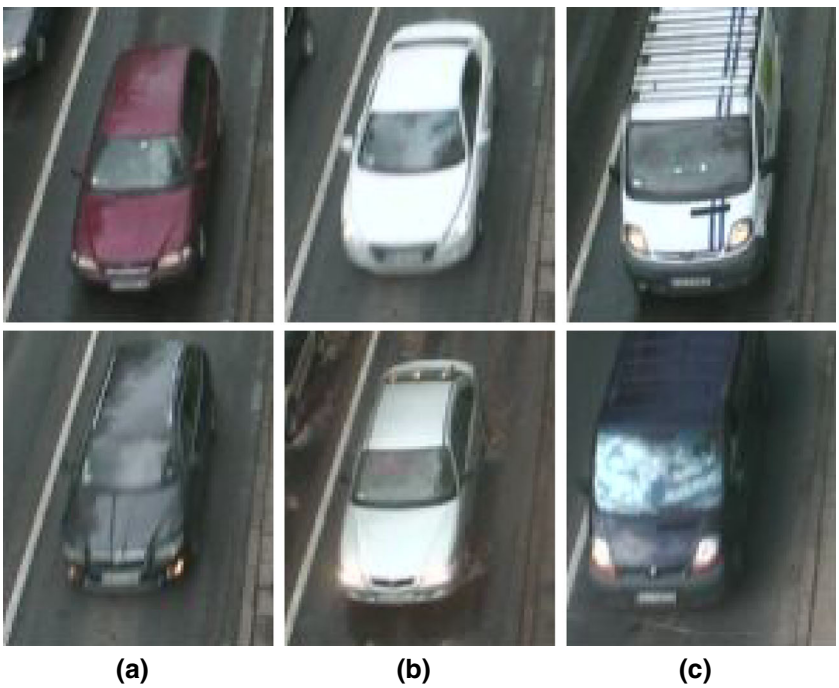

Fig. 17 Car samples: **a** estate, **b** sedan, **c** minibus

Table 8 Detection results of different LBP variants

LBP var.	Block size	Acc. (train) (%)	Acc. (test) (%)
LBP	32	100	81
LBP	16	100	71
LBP	8	100	66
ULBP	32	100	60
NRULBP	32	100	71

In the initial tests the impact on accuracy of the following parameters was evaluated:

- threshold determining method for LBP method,
- LBP variant: basic LBP, ULBP, NRULBP,
- block size: $8 \times 8, 16 \times 16, 32 \times 32$,
- C parameter of the SVM algorithm (penalty weight in the soft-margin SVM).

For the used test set the following impact of parameters on the classification was observed. For different threshold selection methods no significant differences were noticed. Usually, the best result was obtained with threshold as central value.

The block size has direct impact on the final feature vector size. For example, the typical LBP (256 possible values) with an 8×8 block results in 143,360 features, a 16×16 in 35,840 features, and a 32×32 in 8960 features. In the experiment the best results were obtained for the variant 32×32.

A similar impact on the feature vector size has the used LBP variant. For block size 32×32 and ULBP the feature vector is 2065, and for NRULBP 1050. However, the use of these variants results in deterioration of classification performance. The C parameter of the SVM classifier was set to $C = 100$. Results of the performed experiments are summarized in Table 8.

On that basis, the following algorithm option was selected: NRULBP block size 32×32, central pixel value used as threshold, $C = 100$ which gave 71 % accuracy on the test set. The LBP option with higher accuracy required a much longer feature vector and was therefore not considered. It is worth noting that neither the selection of best parameters nor the design of a reliable recognition system were the primary objective of the presented study. The proposed subsystem only demonstrates the possibilities of heterogeneous computing platform in realising this task. In future research, the experiments should be repeated on an extensive database. Also a vehicle size and position normalization procedure should be considered.

Hardware implementation The proposed vehicle type recognition subsystem consists of two main modules.

In the first, the feature vectors are calculated, i.e. the corresponding LBP histograms. This module is separate for each lane. The 160×224 pixel window is divided into 35 blocks. For each block, the maximal histogram value equals 32×32 and, therefore, it requires 11 bits (2^{11}). In total $30 \times 35 \times 11 = 11{,}550$ bits are needed.

For histogram computation the best solution is the *Dual Port Block RAM* (BRAM) available in the reconfigurable logic. Then, one port is used for data read and the second for write operations. For the above parameters, it is required to use 2 BRAM memory blocks for a single lane.

The scheme of the proposed solution is presented in Fig. 18. The module has three operation modes. The first is histogram calculation. The required memory address is

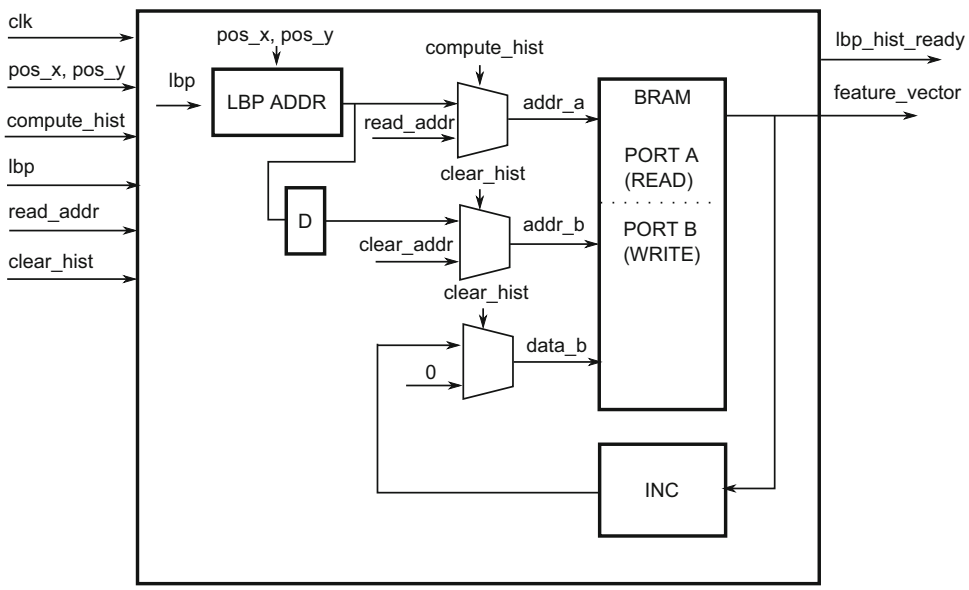

Fig. 18 LBP feature generation module (description in text)

Table 9 LBP feature histogram for a single lane resource usage

Resource	Used	Available	Percentage (%)
FF	49	106,400	0
LUT	128	53,200	0
BRAM	2	140	1

determined using the `lbp` value and pixel to block assignment (determined using `pos_x` and `pos_y`)—module `LBP ADDR`. Then the value stored in RAM is read and incremented—module `INC`. Due to the BRAM memory, where read and write operations are delayed by one clock cycle to instructions, it is necessary to implement additional logic in the `INC` module. This should handle the situation, when consecutive `lbp` values are equal. The new value (incremented) is stored at the same address in the memory (the required address delay is done in module `D`). After the histogram is generated, the flag `lbp_hist_ready` is set to one.

The second mode is histogram read (i.e. feature vector read). In this case, the address is generated outside the module (`read_addr`) and data are transmitted to `feature_vector`. The third mode is histogram reset `clear_hist`—values of all cells are set to 0 (`clear_addr`—internally generated address counter). Resource utilization for a single module is given in Table 9. The estimated maximum frequency equals 267 MHz after synthesis and 177 MHz after place and route.

The second module is the realization of the SVM classifier. Due to the specificity of the application, i.e. the classification is performed only when a vehicle was detected, it is possible to use only one classifier module. The scheme is presented in Fig. 19.

Table 10 Vehicle type classification resource usage

Resource	Used	Available	Percentage (%)
FF	85	106,400	0
LUT	101	53,200	0
DSP 48	3	220	3
BRAM	3	140	1

The state machine (`CONTROLLER`) analyses signals from feature computation modules (`lbp_hist_ready`). If a histogram is ready, then the classification is launched (if there are more than one signals, then the classifications are executed sequentially). In the case of SVM the classification process is rather simple. The feature vectors are multiplied by weights (stored in three BRAMs) and the result is accumulated (`MAC` modules). Finally the bias `b` is added (compare Eq. (12)) in the `ADD_b_X` modules. From the three obtained values `r`, the greatest is selected and if it exceeds a predetermined threshold, then the vehicle belongs to the particular class (`MIN TH`). Resource utilization of the described module is presented in Table 10. The estimated maximum frequency equals 563 MHz after synthesis and 394 MHz after place and route.

7.3 Vehicle colour classification

The performed scientific literature analysis, as well as preliminary experiments, revealed reliable vehicle colour recognition to be a very difficult issue. This is due the following factors: high lightening variability (time of the day, weather conditions), uneven lightning of the vehicle, light reflections, as well as the large variety of car colours used by manufacturers. Also the impact of shadows should

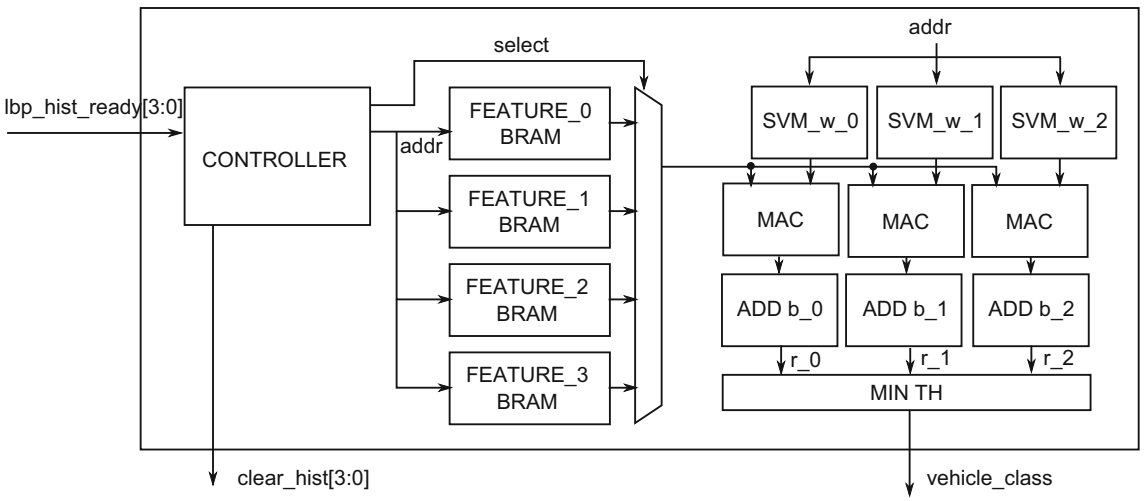

Fig. 19 Vehicle type classification module (description in text)

be mentioned, because often a whole side of the car can be shaded.

When designing a colour recognition module, two factors should be taken into account: the sample acquisition procedure and the used colour model (colour space and representation).

Sample acquisition The first step in the colour samples acquisition procedure is to obtain the vehicle silhouette. This can be done on the basis of edge analysis or so-called background subtraction.

In the first approach it is assumed that all edges between the roadway and the car are quite distinct. Unfortunately, in many cases this is not entirely true and the segmentation requires complex edge analysis, which is not suitable for a pipeline vision system. On the other hand, background subtraction in a proven method for foreground object segmentation. However, as discussed in Sect. 6, the situation at an intersection is quite specific—the background (i.e. road surface) is not visible for long periods and standard approaches tend to fail.

In the described vision system, the second approach was used. However, the model is updated periodically. It is done, when in the given ROI only the road surface is visible. To detect presence of vehicles, an approach similar to the described in Sect. 5 is used.

The background model is stored in the BRAM memory. The *Simple Dual Port RAM* is used, wherein one port is available for writing and one for reading. The update occurs in the case where no vehicle is detected in the ROI. The memory is read only in the case when a car is detected (*enable* flag). This allows to avoid unnecessary operations for every frame.

The model can be stored as colour or greyscale. The first solution requires 3 times larger memory resources. If the maximal ROI size is assumed as 160×224, then for greyscale (8-bits per pixel) 9 BRAM and for RGB colour (24-bits per pixel) 22 BRAM modules are required. Due to limited resources on the target platform XC7Z020 (in total 140 BRAMS) the greyscale variant was used.

In the next stage, the foreground object mask is calculated. It is obtained by thresholding the absolute difference between the current pixel and the background model. In the experiments the threshold was set to 20. An example is shown in Fig. 20.

A comment is required on the area marked as green. The background update mechanism has some delay, which may result in the presence of small parts of cars in the model. Therefore, during colour recognition, the upper area of the ROI is not analysed.

In addition, only pixels on a given lane are considered (cf. mask in Fig. 20c). Because the used sample video sequences were obtained from a camera that was not placed directly over the road but slightly to the side, it caused the omission of certain pixels in the analysed car images. However, for a typical setup, where the camera is mounted exactly above the lane, this problem will not occur or will be marginal.

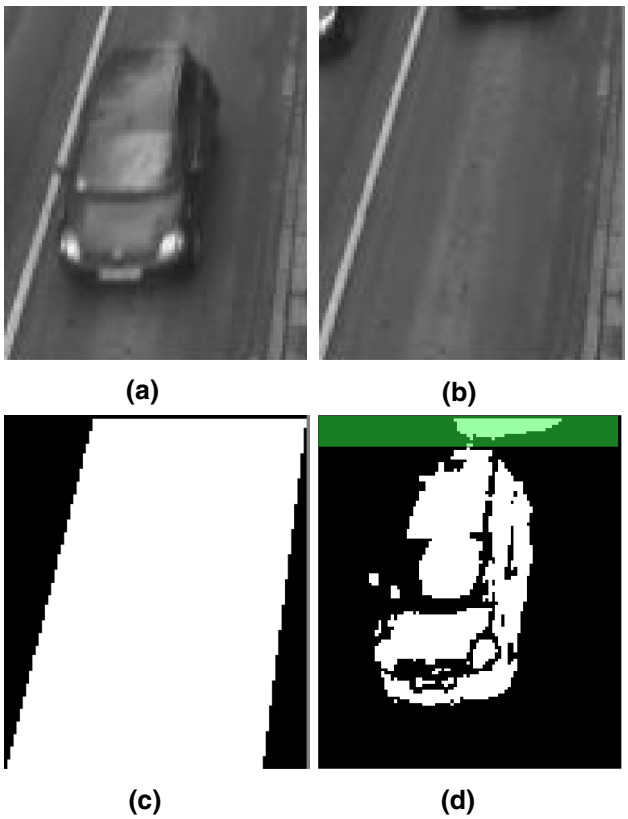

Fig. 20 Foreground object detection example: **a** current frame ROI, **b** corresponding background model, **c** mask for a particular lane, **d** object mask

Analysis of the mask presented in Fig. 20d reveals three problems related to vehicle colour recognition. The first are shadows around the vehicle, especially in the front and on the left side. Unfortunately, despite huge research effort, there are no simple methods of detecting such areas. The authors used colour and texture information, but with only moderate results in previous research. In this example of vision system, the shadow is treated as a separate colour and excluded from analysis using a simple heuristic (described below).

A similar difficulty is caused by shadows present on the car (on the used test sequences on its right side). It is well segmented, in the example it differs even more from the background than other car elements. However, samples in this area can be misinterpreted in the colour classification procedure. This issue requires further research.

The third problem is caused by windscreens. Regardless of the vehicle colour, this area is usually light grey and can

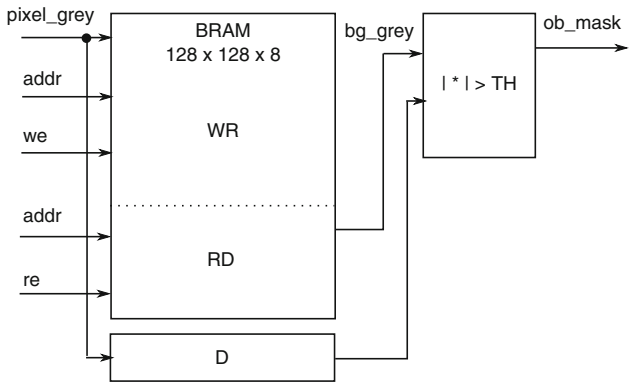

Fig. 21 Foreground object detection module (description in text)

heavily influence the colour recognition. In the proposed solution, a separate colour model for these areas is used. Furthermore, another issue are reflections from the windscreen. They could be misinterpreted as white colour. However, due to they relative small number of such pixels, they had no big impact on the colour recognition.

Hardware implementation The scheme of the proposed foreground object segmentation (in this case cars) is presented in Figure 21. As already stated, the background model for a single lane is stored in dual-port block RAM memory. The write port has the following inputs: `pixel_grey`—pixel value in greyscale, `addr`—address based on the pixel location in the frame, `we`—write enable flag, which is the logic product of: `de`—pixel valid, `inside_bbox`—flag indicating that the pixel is inside a given ROI and `performBackgroundUpdate`—flag indicating that the background model should be updated (set to 1 when in the given ROI no movement and vehicle presence is detected). The read port has inputs: the same address and `re`, flag (`de & inside_bbox & performVehicle Analysis`—set to 1 when a vehicle is detected using the method described in Sect. 7.1). Then, the signal `pixel_grey` is delayed (module `D`) and the absolute difference is computed and thresholded. Finally, the object mask is obtained `ob_mask`. It is then used to gather the colour samples. Resource utilization of the designed module is presented in Table 11. The estimated maximum frequency equals 263 MHz after synthesis and 145 MHz after place and route.

Sample classification When designing a colour classification system, two important decisions should be made. The first relates to the used colour space. The most widely used are: RGB, YCbCr, CIELab or HSV. In this exemplary solution, the YCbCr space is used, as it provides fairly good classification results and the transformation from the RGB space is very simple (unlike, e.g. RGB-HSV conversion).

The second concerns the representation of the colour model. The model could be in the form of:

Table 11 Vehicle silhouette segmentation module for a single lane resource utilization

Resource	Used	Available	Percentage (%)
FF	104	106,400	0
LUT	188	53,200	3
BRAM	9	140	6

- mean values (classification similar to k-means approach),
- Gaussian distributions (independent components),
- Gaussian distributions (dependent components—computationally more complex),
- Gaussian mixture model (GMM),
- three 2D histograms,
- 3D histogram.

In the presented sample application the simplest k-means based approach was used. The recognized colours are black (also shadow), silver, white, red, yellow, green, blue and "windscreen". It should be noted, however, that in a similar way any of the above solutions could be implemented. The Gaussian-based models require appropriate arithmetic and histogram-based larger memory resources (especially the 3D histogram).

The proposed algorithm was evaluated on several test sequences (covering over 30 min). The total number of car samples was 300. The colour recognition accuracy was at the level of 88 %. The errors resulted mainly from incorrect segmentation, especially due to the presence of the shaded areas. This issue should be certainly addressed in future work.

Hardware implementation To optimize the resource usage one colour classification module is used. This is possible because the pixels are always in distinct locations and, therefore, no risk of conflict exists (simultaneous access to the same modules). The scheme of the module is presented in Fig. 22.

Input signals are: `ce`—logical product of foreground object mask, inside ROI flag, analysis flag (i.e. vehicle detection inside the analysis ROI) and `de`, `start_recogni -tion`—signal to start the recognition (product of the analysis and last pixel in ROI flags), `lane_id` – lane label, `pixel_ycbcr`—pixel in YCbCr colour space.

In the first step, the distances between the current sample and all colour models are computed in parallel (modules `DIST`). Subsequently the minimum distance is determined (`min_value`) and the corresponding colour label `min_idx`. The minimum distance is checked against a threshold (module `TH`)—it is possible that a pixel does not correspond to any recognized colour class. In the next step the corresponding counter is incremented. Its selection is a two-step procedure.

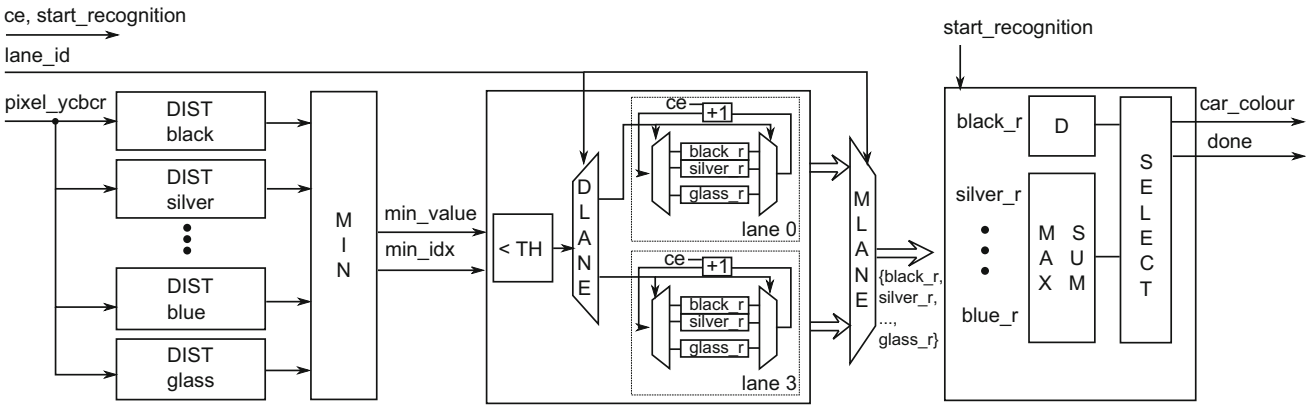

Fig. 22 Vehicle colour recognition module (description in text)

First, the demultiplexer `DLANE` determines the lane (i.e. a particular ROI). Then, a second demultiplexer using `min_idx` determines the required register (e.g. `black_r`). If the signal `ce == 1`, then the register value is incremented.

In the case when the `start_recognition` signal is asserted, the corresponding register bank is selected using the multiplexer `MLANE`. Then, the colours, except black and "windscreen" are added and also the maximum is found—module `MAX SUM`. The "windscreen" colour is omitted in this analysis. The black colour usually also belongs to shadow areas (under the car). This property can be used to implement a simple heuristics (module `SELECT`). First the pixels classified as the "remaining" colours are summed up. If this sum is two times greater than the sum of black pixels, as the recognition result the most common colour among the "remaining" ones is considered. Otherwise, the car is regarded as black. After recognition, the register for the given lane is reset to 0. The output of the module are colour index—`car_colour` and `done` flag, which indicates that the recognition procedure is finished. Resource utilization in presented in Table 12. The estimated maximum frequency equals 157 MHz after synthesis and 97 MHz after place and route.

8 Global modules

In the proposed systems some operations are common to all pixels, and therefore they are implemented as global.

8.1 Consecutive frames differencing

Consecutive frame differencing is used for motion detection. The absolute difference in RGB colourspace is used:

$$dI = |R_i - R_{i-1}| + |G_i - G_{i-1}| + |B_i - B_{i-1}| > \text{th} \qquad (13)$$

Table 12 Vehicle colour classification module resource utilization

Resource	Used	Available	Percentage (%)
FF	657	106,400	0
LUT	1847	53,200	3

Table 13 Consecutive frames differencing module resource usage

Resource	Used	Available	Percentage (%)
FF	79	106,400	0
LUT	116	53,200	0

where RGB_i is the pixel from the frame i, RGB_{i-1} is the pixel from frame $i-1$, a th binarization threshold (constant). The whole operation requires only a few arithmetic operations, the resource usage is presented in Table 13. The estimated maximum frequency equals 357 MHz after synthesis and 222 MHz after place and route.

The size of the image frame in RGB colour space is larger than 1MB. This means that the previous frame cannot be stored in internal BRAM resources, but must be buffered in external DDR3 RAM memory. Thanks to the Zynq device architecture, the memory can be shared between the ARM device running Linux operating system and the FPGA fabric.

Up to four DMA channels can be configured and used by FPGA-based IP cores. The AXI VDMA [69] IP Core provided by Xilinx is employed in the described system. It allows a streaming access to the image frame data, which means that pixels can be stored and read in the same way as they are provided by the camera. The random access to a particular pixel is not allowed, but it is not needed for frame differencing. The estimated (Xilinx EDK tool) resource usage for VDMA IP core is presented in Table 14.

Table 14 VDMA core resource usage

Resource	Used	Available	Percentage (%)
FF	3487	106,400	3
LUT	3151	53,200	6
BRAM	4	140	3

8.2 RGB to YCbCr colour space conversion

The YCbCr colour space is used in the vehicle colour classification. Additionally, the Y component (luminance, brightness) is used in Sobel edge and LBP descriptor modules. The following equation was implemented:

$$\begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 \\ -0.168736 & -0.331264 \\ 0.5 & -0.418688 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix} \quad (14)$$

The required arithmetical operations were realised with the use of hardware DSP modules available in the Zynq device. The resource utilization is presented in Table 15. The estimated maximum frequency equals 707 MHz after synthesis and 429 MHz after place and route.

8.3 LBP 3×3 module

The LBP feature module is based on the delay line approach, which allows to generate a 3×3 context. Then, using the central pixel value and the neighbourhood the LBP is computed. Two LBP outputs are supported: basic LBP and non-redundant uniform LBP. They are used, respectively, for vehicle counting (VDL) and type recognition. The resource usage of the module is summarized in Table 16. The estimated maximum frequency equals 395 MHz after synthesis and 299 MHz after place and route.

8.4 Sobel edge detection module

The Sobel edge detection module is based on the delay line approach, which allows to generate a 3×3 context.

Table 15 RGB to YCbCr module resource usage

Resource	Used	Available	Percentage (%)
FF	101	106,400	0
LUT	78	53,200	0
DSP 48	9	220	4

Table 16 LBP feature module resource usage

Resource	Used	Available	Percentage (%)
FF	66	106,400	0
LUT	75	53,200	0
BRAM	1	140	0

Table 17 Sobel edge detection module resource usage

Resource	Used	Available	Percentage (%)
FF	423	106,400	0
LUT	388	53,200	0
BRAM	3	140	0

Table 18 Gaussian filtering module resource usage

Resource	Used	Available	Percentage (%)
FF	402	106,400	0
LUT	271	53,200	0
BRAM	6	140	2

Then, convolution with two kernels vertical and horizontal is performed. The operation is performed separately for R, G and B colour components and then the results are integrated. Finally, the sum of absolute values of the computed gradients are compared with a fixed threshold. The resource usage of the module is summarized in Table 17. The estimated maximum frequency equals 564 MHz after synthesis and 259 MHz after place and route.

8.5 Gaussian filtering

The Gaussian filtering module for RGB frame (each colour component independently) is also based on the delay line approach. A convolution with a 3×3 kernel is performed. The resource usage of the module is summarized in Table 18. The estimated maximum frequency equals 404 MHz after synthesis and 312 MHz after place and route.

8.6 Position computing module

The module allows to calculate the pixel position in the frame using synchronization signals `de`, `h_sync`, `v_sync` of the video stream and resolution of the analysed image. In addition, it generates a flag `eof`, which indicates the end of the frame. The resources usage is shown in Table 19. The estimated maximum frequency equals 404 MHz after synthesis and 380 MHz after place and route.

Table 19 Position computing module resource usage

Resource	Used	Available	Percentage (%)
FF	21	106,400	0
LUT	48	53,200	0

Table 20 Resource utilisation for the hardware part of the image processing algorithm

Resource	Used	Available	Percentage (%)
FF	8678	106,400	8
LUT	9244	53,200	17
DSP 48	99	220	45
BRAM	60	140	42

9 System integration and evaluation

The described in the previous sections vision system has been implemented on the Zynq SoC device (XC7Z020 CLG484 -1 AP SoC) available at the ZC 702 evaluation board made by Xilinx. In Table 20 the resource utilization for the hardware part of the described image processing system is presented (without communication logic). The estimated maximum frequency equals 261 MHz after synthesis and 166 MHz after place and route.

Two rows of this table require additional comments. Firstly, the resource utilization of DSP 48 modules is quite high. This results from three reasons: the multiplications used in RGB to YCbCr conversion, coefficients scaling in a single VDL and logic designed to determine the current ROI in the queue length estimation subsystem. These operations should be further optimized in future work. Secondly, the internal BRAM resource usage is also quite high. It is the consequence of storing parts of the image, both in a single VDL, as well as in vehicle segmentation module (for vehicle colour recognition). Particularly in the latter case, a modification of the design should be considered. The used 160×224 pixels ROI could be reduced for example by a factor of 2 (e.g. 80×112). This would also simplify the realization of vehicle colour and type recognition modules.

In Table 21 the resource utilisation for the entire hardware system is presented. It includes the designed image processing module, as well as HDMI video signal input/output, AXI bus communication (PL-PS) and external RAM communication (VDMA). The estimated maximum frequency equals 141 MHz after place and route.

Analysis of the presented data leads to some general conclusions. First, the maximum operating frequency of the designed vision system reaches 140 MHz. This is more than enough to enable real-time processing for the target resolution, i.e. 720×576 @ 50 fps (pixel clock 27 MHz). It is also possible to process more than 250 frames of this resolution per second. However, this information is rather theoretical. In practise, such video sources are not used, especially for ITS vision system where there is no need for high frame rate analysis. Much more interesting would be the possibility of processing a high-definition video stream (1920×1080 @ 50 fps—148 MHz pixel clock). After some modifications and optimizations this should be possible and is undoubtedly an interesting direction for further research.

Table 21 Resource utilisation for the hardware part of the entire system

Resource	Used	Available	Percentage (%)
FF	16,591	106,400	15
LUT	16,029	53,200	30
DSP 48	99	220	45
BRAM	101	140	72

Secondly, even in a relatively small device Z-7020 a system for a three or four lane intersection can be implemented. This should be sufficient for many real-life applications. Besides, the resource usage could be further optimized or a slightly bigger device used (Z-7030, Z-7035).

9.1 PL-PS communication

The data transfer between the two parts was handled by the AXI bus. Particularly, the pixels from the VDL, values of three similarity measures *SAD*, *dSX* and *dCEN*, queue length on a lane and vehicle type, as well as colour recognition results were transmitted. Each VDL data were stored in a separate FIFO buffer. Both the FIFOs data lines and control signals (empty and read flags) are mapped in the ARM system as registers in the processor memory space. The buffers state was pooled by the operating system driver (to check if there are data available) and the data from non-empty buffers were transferred to operating system memory space. The used PetaLinux OS enables multitasking, and therefore this solution is not affecting the whole system performance. Furthermore, the patch analysis thread should wait until next data arrives (another patch is ready for analysis). The rest of the OS is working independently and, for example, it is possible to communicate with the system at the same time.

In the software part the PetaLinux operating system was used. The communication with the hardware part was realised with an AXI bus driver. The entire analysis was performed in a single user application.

Tests showed that the communication between the elements of the system works correctly, and the ARM processor was efficient enough to analyse patches and perform other tasks in real time. In the first test scenario sixteen, 64 pixel wide VDL (32 bit data) were evaluated. No processing delays were observed in the ARM core. In the second test, the maximal data transfer via the AXI bus was estimated. A maximum throughput of 27 MB per second was obtained. During this experiment the ARM was only responsible for capturing the data stream (no computations).

9.2 Comparison with similar hardware solutions

Comparison of the proposed solution with others reported in the literature is possible on two levels: the accuracy of the algorithms and parameters of the hardware system. Unfortunately, in both cases the issue is quite difficult. First, as mentioned in Sect. 3, no single ITS video database exists. Therefore, a fair accuracy benchmark is very difficult and time consuming, as it requires the implementation of all the considered algorithms. Besides, the presented in this article algorithmic solutions are only exemplary, although fully functional. Their main aim is to demonstrate the possibilities of a hardware–software architecture.

Secondly, to our best knowledge, the presented here ITS solution is the only one for the Zynq platform (beginning of 2016). Other similar systems have been implemented on various FPGA devices (from Xilinx and Altera). In such a case, the resource utilisation comparison is not fully adequate, as with advances in technology the devices change their parameters (e.g. LUT size). The only simple and informative parameters are: the used image processing algorithms and the supported video stream (resolution, fps), which are summarized in Table 22. It clearly shows that the proposed system is superior to other proposals in terms of supported functionalities and video stream parameters.

9.3 Comparison with software model

An experiment involving the comparison of the proposed hardware–software system with a software application was also conducted. The software model (i.e. an application with the same functionalities as the hardware system) was implemented in C++ programming language based on the OpenCV image processing library [46]. Two platforms were used: a standard PC with Intel i7-4790 quad-core processor @ 3.6 GHz and an ARM Cortex-A9 @ 866 MHz processor available in the considered Zynq device. In both cases the GCC compiler was used.

It should be emphasized that the presented results are only indicative. On one hand, for the image processing module implemented in PL, a performance of 50 frames per second for 720×576 resolution was assumed. The actual maximum performance of the module was not verified experimentally, as it would require a video source capable of providing more than 50 fps. However, the estimated maximal clock frequency of the integrated system (140 MHz) indicates that HD resolution support should be possible after some optimizations.

On the other hand, for the software implementation it is also possible to achieve significant improvements. However, this would require a number of optimizations. The software model was designed as a functional model of the used algorithms and a reference for the hardware modules. Therefore, during its implementation no multi-threading or special instruction sets were used (i.e. MMX, SSE). Additionally, some operations that are implemented in the hardware model for the entire image frame (motion detection, edge detection) could be realized only for some parts of the image (in particular ROIs). This would allow to increase the overall performance.

During the evaluation the average number of frames (720×576 pixels) processed in one second was measured. A 400 images test sequence was used. On the Intel i7 processor the release mode and maximum speed optimization of the GCC compiler were used (Debain 8.2 OS

Table 22 ITS algorithm implemented in FPGA/SoC devices

Work	Functions	Video stream parameters	Device
[21]	Vehicle detection, counting	786×576 @ 25 fps	Virtex II (XC2V6000-4)
[38]	Vehicle detection, counting	Not provided	Cyclone II (not provided)
[9]	Vehicle detection, counting, classification	Not provided	Virtex 4 (XC4VFX60)
[57]	Vehicle detection, counting	128×128 @ 32 fps	Virtex 4 (XC4VLX60)
[67]	Vehicle detection, counting	128×128 @ 32 fps	Virtex 4 (XC4VLX60)
The proposed system	Vehicle detection, counting, speed estimation, classification (type, colour), queue length estimation	720×576 @ 50 fps	Zynq SoC (XC7Z020-1)

Table 23 Comparison of the proposed system with software model performance

Platform	FPS
HW/SW system	Min. 50
Intel i7-4790 @ 3.6 GHz	43
ARM Cortex-A9 @ 866 MHz	3.5

was used). On the ARM processor similar parameters were used. The results are summarized in Table 23.

It can be assumed that further improvement of the i7 implementation would allow to obtain real-time processing for a standard resolution (i.e. 720×576 @ 50 fps), but for HD stream it would be rather a significant challenge. However, such a solution is very energy inefficient and virtually unacceptable in embedded systems (only the considered processor can consume up to 84 W). For the ARM processor a 3.5 frames per second processing speed was obtained. It does not seem likely that even the most advanced optimizations and use of both processor cores would allow to achieve the desired processing capability, even for the standard resolution.

Additionally, a profiling of the software model was performed. The `GNU gprof` tool [22] was used. The results showed that the most time-consuming operations are those carried out for the whole image frame, i.e. LBP descriptor, Sobel edge detection or consecutive frame subtraction. In addition, quite a lot of calculations are required for operations related to queue length estimation or vehicle type or colour recognition. It is worth noting that the patch analysis (the part of the algorithm implemented on the ARM core in the proposed system) is not high in the GPP usage ranking.

The presented results confirm the validity of the assumptions described in Sect. 2 and then realised in the proposed hardware–software system. Implementing the a large part of the application in PL is fully justified, as it easily allowed to obtain real-time image processing for a 720×576 @ 50 fps video stream. On the other hand, it also has some drawbacks. The most important one is the fairly time-consuming process of implementation, testing and running individual hardware modules, as well as of the whole system. However, in the case of embedded systems, particularly used on a large scale (traffic analysis at intersections, smart-cities, etc.), this approach seems quite reasonable. It is worth mentioning that the fine-tuning of a reliable software solutions also requires careful code analysis and testing in terms of reliability, even when writing purely functional code is relatively little time consuming. Of course, a further analysis of the design to determine the optimal HW/SW partitioning is possible and should be considered as a separate future research possibility.

10 Future work

The main goal of this study was to develop a system largely implemented in programmable resources of the Zynq device. Nevertheless, on the occasion, an overview, detailed analysis and selection of algorithms used in earlier works has been done. These solutions have been improved and adapted so that they could be implemented in a hardware–software system. This analysis also pointed out some shortcomings of the described algorithms. From the perspective of system implementation in Zynq, the selection can be seen as illustrating the potential of the system.

Therefore, it is possible to continue their improvement, and even replace some of them with more effective approaches. This especially concerns vehicle type and colour recognition, which is very challenging. For example, instead of LBP features, the HOG could be used. In addition, it seems necessary to propose better algorithms for the analysis of patches to improve the overall accuracy. Proper identification of large vehicles (buses, trucks) makes serious difficulties in the current version of the algorithm. It is also worth carefully examining all the parameters and thresholds used in the algorithms and make an attempt to optimize them or create good configuration procedures. What is more, the system would be able to perform better if an effective shadow detection and elimination procedure was used.

The system could be further optimized, e.g. by sharing computing resources by modules operating on separate lanes or by replacing some multiplications by add/shift approximations. Moreover, the dual-core ARM processor system could be used for more advanced image processing tasks, data analytic, storage and presentation. Different OS configurations described is Sect. 2 should be considered, e.g. the use of both ARM cores.

One possibility is to run Linux OS on one and a real-time operating system on the second core. In this configuration the following computing task division could be applied. The Linux OS would execute high-level image analysis and recognition routines, as well as statistic and communication with the smart camera (e.g. via Ethernet)—in a similar manner as in the above-described version. The ROTS (e.g. the supported FreeRTOS) would be used to directly control the traffic lights—mainly using information about vehicle presence and queue length (both computed in PL). This solution would certainly improve the reliability of the proposed vision system.

New functionalities could also be added to the system. These could involve: abnormal situation detection (collisions, breakdowns, wrong driving direction), vehicle tracking, red light crossing detection, licence plate recognition or vehicle make and model recognition. However, the latter two would require a high-quality, high-resolution image. Therefore, HD or even UltraHD video stream processing should also be considered in further research. Also the topic of optimal HW/SW implementation of the algorithms seems to be very interesting and should be considered in future research. Furthermore, a camera calibration and perspective correction module could also be included to the system.

What is more, new SoC device generations (e.g. Zynq UltraScale+ MPSoC) could allow to significantly improve and extend the proposed system. These devices contain more logic resources, a new internal RAM memory resources, H.265/H.264 hardware video codec, a quad-core ARM processor, a dual-core ARM processor and GPU. Therefore, more advanced and computing intensive ITS algorithms could be implemented.

11 Summary

The article demonstrates the usefulness of heterogeneous Zynq SoC to build an embedded vision system for a smart camera dedicated to traffic surveillance in ITS. To achieve this, the following issues have been resolved: hardware–software architecture required to control the acquisition of the HDMI video signal to the system, the AXI bus-based communication between the FPGA and ARM processor, AXI bus driver for the PetaLinux system and configuration of the operating system itself. To demonstrate the capabilities of the developed architecture the following algorithms were implemented, tested and evaluated on test sequences: vehicle queue length estimation, vehicle detection, counting and speed estimation, as well as vehicle type and colour recognition.

The algorithms were implemented partially in hardware (FPGA) and software (ARM with PetaLinux) and were positively verified on the ZC 702 platform from Xilinx. The system allows to process 50 frames with a resolution of 720 $\times$ 576 pixels per second. An estimate of the maximum operation frequency of the designed modules, as well as the whole hardware system indicates the possibility of pipelined processing of the video stream at 140 MHz frequency. This leads to the conclusion that after further improvement of the project, it would be fairly quickly possible to achieve real-time processing of a HD stream (i.e. 1920 $\times$ 1080 @ 50 fps).

The obtained results demonstrate that Zynq SoC provide a good basis for the implementation of advanced video algorithms and building smart cameras, as they combine the advantages of reconfigurable circuits and general purpose processor system with Linux OS support.

Acknowledgments The authors would like to thank Mr Pawel Tryba. The results, conclusion and experiments described in his MSc Thesis entitled *Video-based vehicle classification* were very useful when working on the described system. The authors would also like to thank Mr Zbigniew Mikrut, PhD, who has worked with video-based ITS system at our laboratory for several years and has inspired and supported the presented work. Finally, many thanks to reviewers. They valuable comments allowed to greatly improve this paper.

References

1. Albiol, A., Albiol, A., Mossi, J.: Video-based traffic queue length estimation. In: Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on, pp. 1928–1932 (2011). doi:10.1109/ICCVW.2011.6130484
2. Altera. https://www.altera.com/products/soc/overview.html (2015). Accessed 06 Dec 2015
3. Asaidi, H., Aarab, A., Bellouki, M.: Shadow elimination and vehicles classification approaches in traffic video surveillance context. J. Vis. Lang. Comput. **25**(4), 333–345 (2014). doi:10.1016/j.jvlc.2014.02.001
4. Athanas, P.M., Abbott, A.L.: Image processing on a custom computing platform. In: Hartenstein, R., Servt, M. (eds.) Field-programmable logic architectures, synthesis and applications, lecture notes in computer science, vol. 849, pp. 156–167. Springer, Berlin Heidelberg (1994). doi:10.1007/3-540-58419-6_86
5. Badura, S., Foltan, S.: Advanced scale-space, invariant, low detailed feature recognition from images—car brand recognition. In: Computer Science and Information Technology (IMCSIT), Proceedings of the 2010 International Multiconference on, pp. 19–23 (2010). doi:10.1109/IMCSIT.2010.5679924
6. Baek, N., Park, S.M., Kim, K.J., Park, S.B.: Vehicle color classification based on the support vector machine method. In: Huang DS, Heutte L, Loog M (eds) Advanced Intelligent Computing Theories and Applications. With Aspects of Contemporary Intelligent Computing Techniques, Communications in Computer and Information Science, vol. 2, pp. 1133–1139. Springer, Berlin Heidelberg (2007). doi:10.1007/978-3-540-74282-1_127
7. Bailey, D.G.: Design for Embedded Image Processing on FPGAs. Wiley (Asia) Pte Ltd, Singapore (2011).
8. Belbachir, A.N.: Smart Cameras. Springer, USA (2010)
9. Bowen, F., Lee, E., Du, E.: A scalable FPGA vehicle monitoring and classification architecture. In: International Conference on Engineering of Reconfigurable Systems and Algorithms, ERSA (2011)

10. Camellini, G., Felisa, M., Medici, P., Zani, P., Gregoretti, F., Passerone, C., Passerone, R.: 3DV an embedded, dense stereo-vision-based depth mapping system. In: Intelligent Vehicles Symposium Proceedings, 2014 IEEE, pp. 1435–1440 (2014). doi:10.1109/IVS.2014.6856563
11. Cao, J., Li, L.: Vehicle objects detection of video images based on gray-scale characteristics. In: Education Technology and Computer Science, 2009. ETCS '09. First International Workshop on, vol. 2, pp. 936–940 (2009). doi:10.1109/ETCS.2009.472
12. Chen, L.C., Hsieh, J.W., Yan, Y., Chen, D.Y.: Vehicle make and model recognition using sparse representation and symmetrical SURFs. Pattern Recognit. **48**(6), 1979–1998 (2015). doi:10.1016/j.patcog.2014.12.018
13. Chen, Z., Pears, N., Freeman, M., Austin, J.: Road vehicle classification using support vector machines. In: Intelligent Computing and Intelligent Systems, 2009. ICIS 2009. IEEE International Conference on, vol. 4, pp. 214–218 (2009). doi:10.1109/ICICISYS.2009.5357707
14. Crookes, D., Benkrid, K.: FPGA implementation of image component labeling. In: Proceedings. SPIE 3844, Reconfigurable Technology: FPGAs for Computing an Applications (1999). doi:10.1117/12.359538
15. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on, vol. 1, pp. 886–893 (2005). doi:10.1109/CVPR.2005.177
16. Dobai, R., Sekanina, L.: Image filter evolution on the Xilinx Zynq Platform. In: Adaptive Hardware and Systems (AHS), 2013 NASA/ESA Conference on, pp. 164–171 (2013). doi:10.1109/AHS.2013.6604241
17. Ghasemi, A., Safabakhsh, R.: A real-time multiple vehicle classification and tracking system with occlusion handling. In: Intelligent Computer Communication and Processing (ICCP), 2012 IEEE International Conference on, pp. 109–115 (2012). doi:10.1109/ICCP.2012.6356172
18. Glowacz, A., Mikrut, Z., Pawlik, P.: Video detection algorithm using an optical flow calculation method. In: Dziech A, Czyzewski A (eds) Multimedia Communications, Services and Security, Communications in Computer and Information Science, vol. 287, pp. 118–129. Springer, Berlin Heidelberg (2012). doi:10.1007/978-3-642-30721-8_12
19. Gorgon, M.: VLSI based pipeline architecture for real time image preprocessing. In: Proceedings of the 5th School of Computer Vision and Graphics, pp. 233–239. Wydawnictwo Format, Wroclaw (1994)
20. Gorgon, M.: Parallel performance of the fine-grain pipeline FPGA image processing system. Opto-Electron. Rev. **20**(2), 153–158 (2012). doi:10.2478/s11772-012-0021-2
21. Gorgon, M., Pawlik, P., Jablonski, M., Przybylo, J.: Fpga-based road traffic videodetector. In: Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on, pp. 412–419 (2007). doi:10.1109/DSD.2007.4341500
22. GPROF https://sourceware.org/binutils/docs/gprof/ (2015). Accessed 28 Dec 2015
23. Han, T., Liu, G.W., Cai, H., Wang, B.: The face detection and location system based on Zynq. In: Fuzzy Systems and Knowledge Discovery (FSKD), 2014 11th International Conference on, pp. 835–839 (2014). doi:10.1109/FSKD.2014.6980946
24. Hsieh, J.W., Yu, S.H., Chen, Y.S., Hu, W.F.: Automatic traffic surveillance system for vehicle tracking and classification. Intell Transp Syst IEEE Trans **7**(2), 175–187 (2006). doi:10.1109/TITS.2006.874722
25. Hsieh, J.W., Chen, L.C., Chen, D.Y., Cheng, S.C.: Vehicle make and model recognition using symmetrical SURF. In: Advanced Video and Signal Based Surveillance (AVSS), 2013 10th IEEE International Conference on, pp. 472–477 (2013). doi:10.1109/AVSS.2013.6636685
26. Hsieh, J.W., Chen, L.C., Chen, S.Y., yu Chen D, Alghyaline, S., Chiang, H.F.: Vehicle color classification under different lighting conditions through color correction. Sens. J. IEEE **15**(2), 971–983 (2015). doi:10.1109/JSEN.2014.2358079
27. Jablonski, M., Gorgon, M.: Handel-C implementation of classical component labelling algorithm. In: Digital System Design, 2004. DSD 2004. Euromicro Symposium on, pp. 387–393 (2004). doi:10.1109/DSD.2004.1333301
28. Kapela, R., Gugala, K., Sniatala, P., Swietlicka, A.: Kolanowski K (2015) Embedded platform for local image descriptor based object detection. Appl. Math. Comput. **267**, 419–426 (2015). doi:10.1016/j.amc.2015.02.029 **(the FourthEuropean Seminar on Computing (ESCO 2014))**
29. Kim, K.J., Park, S.M., Choi, Y.J.: Deciding the number of color histogram bins for vehicle color recognition. In: Asia-Pacific Services Computing Conference, 2008. APSCC '08. IEEE, pp. 134–138 (2008). doi:10.1109/APSCC.2008.207
30. Komorkiewicz, M., Kluczewski, M., Gorgon, M.: Floating point HOG implementation for real-time multiple object detection. In: Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on, pp. 711–714 (2012). doi:10.1109/FPL.2012.6339159
31. Komorkiewicz, M., Kryjak, T., Gorgon, M.: Efficient hardware implementation of the Horn-Schunck algorithm for high-resolution real-time dense optical flow sensor. Sensors **14**(2), 2860 (2014). doi:10.3390/s140202860
32. Kryjak, T., Komorkiewicz, M., Gorgon, M.: FPGA implementation of real-time head-shoulder detection using local binary patterns, SVM and foreground object detection. In: Design and Architectures for Signal and Image Processing (DASIP), 2012 Conference on, pp. 1–8 (2012)
33. Kryjak, T., Komorkiewicz, M., Gorgon, M.: Hardware-software implementation of vehicle detection and counting using virtual detection lines. In: Design and Architectures for Signal and Image Processing (DASIP), 2014 Conference on, pp. 1–8 (2014a). doi:10.1109/DASIP.2014.7115618
34. Kryjak, T., Komorkiewicz, M., Gorgon, M.: Real-time implementation of foreground object detection from a moving camera using the {ViBE} algorithm. Computer Science and Information Systems. **11**(4), 1617–1637 (2014b). doi:10.2298/CSIS131218055K
35. Kunz, M., Ostrowski, A., Zipf, P.: An FPGA-optimized architecture of Horn and Schunck optical flow algorithm for real-time applications. In: Field Programmable Logic and Applications (FPL), 2014 24th International Conference on, pp. 1–4 (2014). doi:10.1109/FPL.2014.6927406
36. Liu, Y., Wang, K.: Vehicle classification system based on dynamic Bayesian network. In: Service Operations and Logistics, and Informatics (SOLI), 2014 IEEE International Conference on, pp. 22–26 (2014). doi:10.1109/SOLI.2014.696068710.1109/SOLI.2014.6960687
37. Llorca, D., Arroyo, R., Sotelo, M.: Vehicle logo recognition in traffic images using HOG features and SVM. In: Intelligent Transportation Systems—(ITSC), 2013 16th International IEEE Conference on, pp. 2229–2234 (2013). doi:10.1109/ITSC.2013.6728559
38. Menezes, G., Silva-Filho, A.: Motion detection of vehicles based on FPGA. In: Programmable Logic Conference (SPL), 2010 VI Southern, pp. 151–154 (2010). doi:10.1109/SPL.2010.5483022
39. MentorGraphics. https://www.mentor.com/products/fpga/handel-c/ (2015a). Accessed 06 Dec 2015
40. MentorGraphics https://www.mentor.com/products/fpga/handel-c/pixelstreams/ (2015b). Accessed 06 Dec 2015

41. Mithun, N., Rashid, N., Rahman, S.: Detection and classification of vehicles from video using multiple time-spatial images. Intelligent Transportation Systems, IEEE Transactions on **13**(3), 1215–1225 (2012). doi:10.1109/TITS.2012.2186128
42. Monson, J., Wirthlin, M., Hutchings, B.: Implementing high-performance, low-power FPGA-based optical flow accelerators in C. In: Application-Specific Systems, Architectures and Processors (ASAP), 2013 IEEE 24th International Conference on, pp. 363–369 (2013). doi:10.1109/ASAP.2013.6567602
43. Nguyen, D.T., Ogunbona, P., Li, W.: Human detection with contour-based local motion binary patterns. In: Image Processing (ICIP), 2011 18th IEEE International Conference on, pp. 3609–3612 (2011). doi:10.1109/ICIP.2011.6116498
44. Ojala, T., Pietikinen, M., Harwood, D.: A comparative study of texture measures with classification based on featured distributions. Pattern Recognit. **29**(1), 51–59 (1996). doi:10.1016/0031-3203(95)00067-4
45. Ojala, T., Pietikainen, M., Maenpaa, T.: Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. Pattern Anal. Mach. Intell. IEEE Trans. **24**(7), 971–987 (2002). doi:10.1109/TPAMI.2002.1017623
46. OpenCV. http://opencv.org (2015). Accessed 31 May 2015
47. Otsu, N.: A threshold selection method from gray-level histograms. Syst. Man Cybernet. IEEE Trans. **9**(1), 62–66 (1979). doi:10.1109/TSMC.1979.4310076
48. Pamula, W.: Object classification methods for application in FPGA based vehicle video detector. Transp. Probl. **4**(2):5–14, (2009)
49. Pan, S., Shi, L., Guo, S., Guo, P., He, Y., Xiao, R.: A low-power SoC-based moving target detection system for amphibious spherical robots. In: Mechatronics and Automation (ICMA), 2015 IEEE International Conference on, pp. 1116–1121 (2015). doi:10.1109/ICMA.2015.7237642
50. Pang, C., Lam, W., Yung, N.H.C.: A method for vehicle count in the presence of multiple-vehicle occlusions in traffic images. Intell. Transp. Syst. IEEE Trans. **8**(3), 441–459 (2007). doi:10.1109/TITS.2007.902647
51. Placzek, B.: A real time vehicle detection algorithm for vision-based sensors. In: Bolc, L., Tadeusiewicz, R., Chmielewski, L., Wojciechowski, K. (eds.) Computer Vision and Graphics, Lecture Notes in Computer Science, vol 6375, Springer, Berlin Heidelberg, pp. 211–218 (2010). doi:10.1007/978-3-642-15907-7_26
52. Pletzer, F., Tusch, R., Boszormenyi, L., Rinner, B.: Robust traffic state estimation on smart cameras. In: Advanced Video and Signal-Based Surveillance (AVSS), 2012 IEEE Ninth International Conference on, pp. 434–439 (2012). doi:10.1109/AVSS.2012.63
53. Rabiu, H.: Vehicle detection and classification for cluttered urban intersection. Int. J. Comput. Sci. Eng. Appl. **3**(1):37–47, (2012)
54. Russell, M., Fischaber, S.: OpenCV based road sign recognition on Zynq. In: Industrial Informatics (INDIN), 2013 11th IEEE International Conference on, pp. 596–601 (2013). doi:10.1109/INDIN.2013.6622951
55. Satzoda, R., Suchitra, S., Srikanthan, T., Chia, J.: Vision-based vehicle queue detection at traffic junctions. In: Industrial Electronics and Applications (ICIEA), 2012 7th IEEE Conference on, pp. 90–95 (2012). doi:10.1109/ICIEA.2012.6360703
56. Schwiegelshohn, F., Hubner, M.: Design of an attention detection system on the Zynq-7000 SoC. In: ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on, pp. 1–6 (2014). doi:10.1109/ReConFig..7032510
57. Szczepanski, S., Wöjcikowski, W., Pankiewicz, B., Klosowski, M., Zaglewski, R.: FPGA and ASIC implementation of the algorithm for traffic monitoring in urban areas. Tech. Sci. Bull. Pol. Acad. Sci. (2011). doi:10.2478/v10175-011-0017-y
58. Vapnik, V.N.: The nature of statistical learning theory. Springer, New York (1995)
59. Velez, G., Corts, A., Nieto, M., Vlez, I., Otaegui, O.: A reconfigurable embedded vision system for advanced driver assistance. JJ. Real-Time Image Process. **10**(4), 725–739 (2015). doi:10.1007/s11554-014-0412-3
60. van der Wal, G., Zhang, D., Kandaswamy, I., Marakowitz, J., Kaighn, K., Zhang, J., Chai, S.: FPGA acceleration for feature based processing applications. In: Computer Vision and Pattern Recognition Workshops (CVPRW), 2015 IEEE Conference on, pp. 42–47 (2015). doi:10.1109/CVPRW.2015.7301365
61. Wang, C., Shi, Z.K.: Design and FPGA implementation of real-time vehicle queue detection system based on image processing. J Transp Syst Eng Inf Technol **12**(3), 65 (2012)
62. Wang, Y.C., Hsieh, C.T., Han, C.C., Fan, K.C.: Vehicle type classification from surveillance videos on urban roads. In: Ubi-Media Computing and Workshops (UMEDIA), 2014 7th International Conference on, pp. 266–270 (2014). doi:10.1109/U-MEDIA.2014.69
63. Weina, L., Haifang, W., Yuquan, M., Lihong, Z., Qingzhu, W.: Image detection to vehicle queue length of crossroad based on DSP. In: Jiang, L. (ed.) Proceedings of the 2011 International Conference on Informatics, Cybernetics, and Computer Engineering (ICCE2011) November 19–20, 2011, Melbourne, Australia, Advances in Intelligent and Soft Computing, vol. 110, pp. 237–243. Springer, Berlin Heidelberg (2012). doi:10.1007/978-3-642-25185-6_32
64. Wiatr, K., Kasperek, P., Rajda, P.: Multiprocessor unit for a fast image data pre-processing in real time applications. In: Proceedings of the 5th School of Computer Vision and Graphics, pp. 339–344. Wydawnictwo Format, Wroclaw (1994)
65. Wieclawek, W., Pietka, E.: Car segmentation and colour recognition. In: Mixed Design of Integrated Circuits Systems (MIXDES), 2014 Proceedings of the 21st International Conference, pp. 426–429 (2014). doi:10.1109/MIXDES.2014.6872234
66. Wojcikowski, M., Zaglewski, R., Pankiewicz, B.: FPGA-based real-time implementation of detection algorithm for automatic traffic surveillance sensor network. J. Signal Process. Syst. **68**(1), 1–18 (2012). doi:10.1007/s11265-010-0569-3
67. Wojcikowski, M., Zaglewski, R., Pankiewicz, B., Klosowski, M., Szczepanski, S.: Hardware-software implementation of a sensor network for city traffic monitoring using the FPGA- and ASIC-based sensor nodes. J. Signal Process. Syst. **71**(1), 57–73 (2013). doi:10.1007/s11265-012-0681-7
68. Wu, Y.T., Kao, J.H., Shih, M.Y.: A vehicle color classification method for video surveillance system concerning model-based background subtraction. In: Qiu, G., Lam, K., Kiya, H., Xue, X.Y., Kuo, C.C., Lew, M. (eds.) Advances in Multimedia Information Processing - PCM 2010, Lecture Notes in Computer Science, vol 6297, Springer, Berlin Heidelberg, pp. 369–380 (2010). doi:10.1007/978-3-642-15702-8_34
69. Xilinx. AXI Video Direct Memory Access v6.2. Xilinx (2015a)
70. Xilinx. http://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html (2015b). Accessed 06 Dec 2015
71. Xilinx. http://www.xilinx.com/products/silicon-devices/soc.html (2015c). Accessed 06 Dec 2015
72. Yang, D., Xin, L., Chen, Y.: A robust detection method of vehicle queue and dissipation during evening rush hour. In: Electric Information and Control Engineering (ICEICE), 2011 International Conference on, pp. 1104–1107 (2011). doi:10.1109/ICEICE.2011.5778067
73. Yang, M.T., Jhang, R.K., Hou, J.S.: Traffic flow estimation and vehicle-type classification using vision-based spatial-temporal profile analysis. Comput. Vis. IET **7**(5), 394–404 (2013). doi:10.1049/iet-cvi.2012.0185

74. Yao, Y., Wang, K., Xiong, G.: Embedded technology and algorithm for video-based vehicle queue length detection. In: Service Operations and Logistics, and Informatics (SOLI), 2013 IEEE International Conference on, pp. 45–50 (2013). doi:10.1109/SOLI.2013.6611379
75. Yingfeng, C., Weigong, Z., Hai, W.: Measurement of vehicle queue length based on video processing in intelligent traffic signal control system. In: Measuring Technology and Mechatronics Automation (ICMTMA), 2010 International Conference on, vol. 2, pp. 615–618 (2010). doi:10.1109/ICMTMA.2010.354
76. Zanin, M., Messelodi, S., Modena, C.: An efficient vehicle queue detection system based on image processing. In: Image Analysis and Processing, 2003. Proceedings. 12th International Conference on, pp. 232–237 (2003). doi:10.1109/ICIAP.2003.1234055
77. Zhang, Y., Xu, M., Shen, H.: Design of face detection system based on FPGA. In: Tan, Y., Shi, Y., Mo, H. (eds.) Advances in Swarm Intelligence, Lecture Notes in Computer Science, vol. 7929, pp. 404–410. Springer, Berlin Heidelberg (2013). doi:10.1007/978-3-642-38715-9_48

Tomasz Kryjak received MSc degree in Automatics and Robotics in 2008 and Ph.D. degree in 2013 both from AGH University of Science and Technology in Krakow, Poland. From 2008 onwards he has a permanent position at the Department of Automatics and Biomedical Engineering AGH-UST, currently Assistant Professor. His current research is focused on image processing, analysis and recognition, especially advanced video surveillance systems, advanced driver assistance systems and ITS vision system, reconfigurable FPGA systems, hardware acceleration of vision algorithms, embedded vision systems and software/hardware co-design. He is a member of the IEEE Circuits and System Society He is author or co-author of more than 40 research papers. and one monograph.

Mateusz Komorkiewicz received MSc degree in Automatics and Robotics in 2010 and Ph.D. degree in 2014 both from AGH University of Science and Technology in Krakow, Poland. Since 2015 he works as R&D Engineer at Delphi Automotive. His main research interest areas are vision-based automotive active safety systems and hardware acceleration of vision algorithms using FPGA devices.

Marek Gorgon received MSc, Ph.D. and D.Sc. (habilitation) from AGH University of Science and Technology in Krakow, Poland. From 1994 onwards permanent position at the Department of Automatics and Biomedical Engineering of the AGH-UST, currently Associate Professor. His research interests include: image processing, software–hardware co-design, reconfigurable devices, embedded systems and applications. He is a Senior Member of IEEE, ECSI Member and AGH-UST Senate Member. He serves for the International Program Committees of DASIP, ReConFig and ERSA. He is an author of two monographs, and 90 journal and conference papers and reports.